

NAVAL POSTGRADUATE SCHOOL

Monterey, California



**POLYNOMIAL TRANSFER LOT SIZING
TECHNIQUES FOR BATCH PROCESSING
ON CONSECUTIVE MACHINES**

DAN TRIETSCH

September 1989

Approved for public release; distribution unlimited.

FedDocs
D 208.14/2
NPS-54-89-011

Prepared for:

Naval Postgraduate School
Monterey, CA 93943

DUPLICATE
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93946-002

Fed Docs
D 208 1412
NPS-54-89-011

**NAVAL POSTGRADUATE SCHOOL
Monterey, California**

RADM. R. W. West, Jr.
Superintendent

Harrison Shull
Provost

The research summarized herein was accomplished with resources provided by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:

UNCLASSIFIED

DUDLEY BOOK LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-8002

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release Distribution Unlimited	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE			5 MONITORING ORGANIZATION REPORT NUMBER(S)	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS-54-89-011			7a NAME OF MONITORING ORGANIZATION	
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (If applicable)	7b ADDRESS (City, State, and ZIP Code)	
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O&MN, Direct Funding	
8a NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School		8b OFFICE SYMBOL (If applicable)	10 SOURCE OF FUNDING NUMBERS	
8c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			PROGRAM ELEMENT NO	PROJECT NO
			TASK NO	WORK UNIT ACCESSION NO
11 TITLE (include Security Classification) Polynomial Transfer Lot Sizing Techniques for Batch Processing on Consecutive Machines (UNCLASSIFIED)				
12 PERSONAL AUTHOR(S) Dan Trietsch				
13a TYPE OF REPORT Final Report		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) September 1989
15 PAGE COUNT 48				
16 SUPPLEMENTARY NOTATION				
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	Batch Production; Transfer Lots; Minimal Makespan; MRP; OPT	
19 ABSTRACT (Continue on reverse if necessary and identify by block number) Using transfer lots, we can overlap the processing of a batch on several consecutive machines and thus reduce the makespan considerably. This in turn promotes work-in-process reduction. In this paper we investigate the transfer lots sizing problem for a given batch size under two operating procedures. Our objective is to minimize the makespan subject to a transferring budget. An important part of the solution involves partitioning the problem to subsets of machines without losing optimality. For each part (subset), the first and the last machines operate continuously while intermediate machines may idle intermittently. The first operating procedure we consider calls for the lots to be identical across all machines in each subset. The second operating procedure allows sub-lots for some of the machines or for some of the lots. Though more elaborate, the second operating procedure yields demonstrably superior results. The techniques provide satisficing feasible solutions, which can also serve as efficient bounds for an exact branch and bound integer linear programming model.				
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Dan Trietsch			22b TELEPHONE (include Area Code) 408-646-2456	22c OFFICE SYMBOL 54Tr

**POLYNOMIAL TRANSFER LOT SIZING TECHNIQUES FOR
BATCH PROCESSING ON CONSECUTIVE MACHINES**

by

Dan Trietsch*

September 1989

* Code 54Tr, Naval Postgraduate School, Monterey, CA 93943-5000.

Polynomial Transfer Lot Sizing Techniques for
Batch Processing on Consecutive Machines

Abstract

Using transfer lots, we can overlap the processing of a batch on several consecutive machines, and thus reduce the makespan considerably. This in turn promotes work-in-process reduction. In this paper we investigate the transfer lots sizing problem for a given batch size under two operating procedures. Our objective is to minimize the makespan subject to a transferring budget. An important part of the solution involves partitioning the problem to subsets of machines without losing optimality. For each part (subset), the first and the last machines operate continuously while intermediate machines may idle intermittently. The first operating procedure we consider calls for the lots to be identical across all machines in each subset. The second operating procedure allows sub-lots for some of the machines or for some of the lots. Though more elaborate, the second operating procedure yields demonstrably superior results. The techniques provide satisficing feasible solutions, which can also serve as efficient bounds for an exact branch and bound integer linear programming model.

1. Introduction

In recent years the Japanese have achieved monumental industrial success by implementing the **just-in-time (JIT)** production system on a nationwide scale [9; 4, pp. 736-769]. A basic tenet of JIT is that large batches are contra-productive in more than one way. For instance, they cause excessive work-in-process (**WIP**), excessive lead-time, and reduced flexibility. Large batches also compromise quality, because by the time a defect is detected it is too late to do anything about it. Therefore, JIT calls for small batches, ideally of one unit each.

Other important elements of JIT--beyond the scope of this paper--are total quality management, workers' participation (**Quality Circles**), and striving for constant improvement. Our main concern here is with aspects of materials flow.

Also known as **The Toyota Method**, JIT is designed primarily for the repetitive manufacturing environment. It is a pull system, that is, usage downstream authorizes fabrication upstream. Assembly lines, often found in the repetitive manufacturing environment, are conducive to moving parts one-by-one, as urged by JIT. Parts required for assembly or fabrication are fed to the right stations in small containers. The units in each container usually make up a production batch. To avoid disruptions, buffers comprising a small number of containers are allowed in front of all stations. To avoid excessive WIP, strict limits on the number of containers in each buffer are observed. Part of JIT is a continuous effort to reduce these buffers, and still maintain smooth output.

In the mass production environment there are few potential setups for each machine (we use the term **machine** as a generic for any station where the products have to be processed). To make small batches possible, these setups have to be vigorously streamlined. Reducing setups that used to take several hours to less than 10 minutes is a must under JIT. For an illuminating text on this issue and its impact on the evolution of JIT, see Shingo [16].

In contrast, for medium volume production, and even more so in custom job shops, a large variety of products are produced. Therefore, the number of potential setups increases, and it becomes progressively uneconomical to reduce all of them. Under such circumstances, specifying large batches may be necessary.

Can we capture the major advantages of JIT--such as reducing the lead-time and the WIP inventory--without setting the machinery up more than once per batch, while still specifying sizable batches? Goldratt, the developer of OPT (Optimized Production Technology) [7; 12, pp. 692-715; 10], answered this question in the affirmative. Although OPT does not live up to its name, it is a

sophisticated production control system that successfully applies many JIT ideas to batch production.

It is possible to adopt the OPT philosophy, also known as **synchronized manufacturing** [4, pp. 790-839], without using any computerized system. Nevertheless, many perceive OPT as a competitor of MRP (I and II) [13; 12, pp. 655-658]. Our stance in this paper, following Vollman [20], is that OPT is a potential *enhancement* to MRP. There are four key features in OPT that most MRP packages do not support [10; 20]: (i) concentrating on bottleneck resources; (ii) scheduling activities on bottlenecks (and downstream from bottlenecks) forward instead of backwards, thus utilizing them fully; (iii) specifying WIP buffers (only) in strategic locations (for example, in front of bottlenecks); and, (iv) allowing transfer lots to be smaller than the batches they belong to, thus overlapping the processing on sequential machines.

It is the transfer lots scheme that yields the major lead-time and WIP reductions that OPT achieves. According to a broad interpretation of the OPT principles, these lots need not necessarily be of equal size. Judging by the output of OPT, however, it seems that they do use transfer lots of constant size [10]. (In this paper we allow the lots to vary.)

Although our main concern here is with (iv) and not with (i) through (iii), we note that the literature on scheduling is oriented to forward-scheduling, so it applies to scheduling bottlenecks [e.g., 3; 5]. Also note that linear programming can be used not only to identify bottlenecks (i.e., binding constraints), but also to optimize the product-mix. Ronen [14] gives an analytic model for (iii), based on the newsboy model. Ronen and Starr [15] discuss the relationship between OPT and well-known optimization methods.

Some work has also been published in the realm of (iv). Recent examples are Graves and Kostreva [8], and Truscott [18; 19]. The interested reader may find references to earlier efforts there. The assumptions in [8] are: (i) constant demand, and (ii) equal production rates for all machines. The model in [8] is developed for two machines. It optimizes the number of lots under the constraint that they should be strictly equal and integral. If more than two machines are involved, the authors apply their model on a pair-by-pair basis. [19] is based on [18], and does not make the constant production-rate assumption. Several stages are investigated under a restriction that once a batch starts on a machine, it is run continuously to completion. Transfers are limited to multiples of equal-sized sub-batches. Limitations on the transportation capacity are also taken into account. [18] and [19] are oriented towards implementation and dwell less on theoretical issues.

Trietsch [17] obtains optimal lots for one batch on two machines. The assumptions are that (i) the units can be transferred one-by-one or in any combination, up to and including the whole batch; (ii) the batch size is given; and (iii) the number of transfers is either constrained by a budget or by a

limitation on the transportation resources (e.g., there are j vehicles available for the transfers, so lot $j+1$ cannot be moved until the first vehicle returns). The solution is then extended to several batches that have to be processed on the same two machines in the same order. Finally, a fast heuristic is introduced to extend the model to several machines on a pair-by-pair basis (similarly to [8; 18; 19]). In the latter case, transferring a lot incurs a cost that may be different for different machines, and there is a budget constraint on the total transferring expenditure. The number and composition of lots may change for each machine, to utilize the budget better.

Let us look at a simple example: we have to process 250 widgets on 4 machines. Machines 1, 2, 3 and 4 take 1, 2, 1 and 3 minutes per widget, respectively. Without splitting the batch to lots, the makespan is 1750 minutes. Suppose now that the budget allows two transfer lots from each machine. If we stipulate equal lots and require each machine to process all the widgets continuously (as in [8], but note that the production rate is not equal), the makespan will be 1375 minutes, a reduction of 21.4% (Figure 1).

 Insert Figure 1 about here

By allowing **intermittent idling** in Machine 3 (as OPT does), while still specifying equal transfer lots (as OPT probably does), we can reduce the makespan further to 1250, a total reduction of 28.6% (Figure 2).

 Insert Figure 2 about here

Instead of allowing intermittent idling, if we allow the lot sizes to vary for each machine, as in [17], we can reduce the makespan to 1230, a total reduction of 29.7% (Figure 3). (See Section 6 for computational details of this and the following illustrations.)

 Insert Figure 3 about here

In this paper we generalize the results of [17] for one batch and several consecutive machines by allowing intermittent idling. As in [17], we also allow varying lot sizes. In the present example this can further reduce the makespan to 1155, a total reduction of 34% (Figure 4).

 Insert Figure 4 about here

A crucial part of the solution involves schemes designed to **partition** the problem without losing optimality. For each part (subset), we constrain the first and the last machines to operate continuously

(see Section 2 for a formal definition of partitions). For instance, Machine 2 in Figure 4 is the last machine of the subset $\{1, 2\}$, and the first machine of the subset $\{2, 3, 4\}$. The first operating procedure we consider calls for the lots to maintain their composition across all machines in each subset, but allows different lots across subsets, as in Figures 3 and 4. The second operating procedure allows the use of **sub-lots**. That is, we distinguish between **parent lots** that remain intact across all machines in each subset as before, and sub-lots that make up the parent lots. Figure 5(b) illustrates such a case, where the first parent lot is one unit, and the second parent lot includes four units. On Machine 1, the second parent lot includes two sub-lots; Machine 2 recombines them to a single lot.

 Insert Figure 5 about here

Given a large enough budget, the first procedure can always achieve any feasible makespan. If necessary, we can do that by transferring the units one-by-one. The second procedure, however, may achieve the same makespan with a smaller budget than the first, and in this sense it is superior.

This paper develops fast satisficing solution algorithms for minimizing the makespan under a transferring budget. The algorithms are easy to program and fast; their worst case complexity is polynomial. They yield feasible integral solutions, and are intended to be implemented in new or existing MRP systems. The algorithms do not require the use of any external mathematical programming packages.

It is also possible to find the minimal makespan by integer linear programming (ILP). When solving by ILP, the satisficing solutions can serve as efficient upper bounds. An ILP model is presented in the Appendix.

Following presentation of an early version of this paper in ORSA/TIMS St. Louis (October 1987), this author became aware that Baker was independently developing a similar model [1]. Baker restricts the lots to retain their composition across all machines (in contrast to retaining their compositions in each subset only here). He solves for two machines and several lots, and for three machines and two lots. His model approximates the solution by relaxing the integrality constraints on the lot sizes. The solution is by a set of rules inspired by a linear programming (LP) formulation. (The same formulation can serve to solve for several machines and several lots).

Baker and Pyke [2] extend the results of [1] to several machines and two transfer lots, under the same assumptions. The solution is achieved by minimizing the maximal path in a network. [2]'s result for the first example would be lots of 107.143 and 142.857. The makespan is 1178.571, or 23.571 more than in Figure 4 (24 when integrality constraints are introduced).

The rest of the paper includes 11 sections. Section 2 introduces the formal problem. Sections 3 through 5 deal with the first operating procedure: Section 3 examines a basic model for the first operating procedure, under the assumption that all intermediary machines can handle the loads assigned to them without requiring a partition of the problem; Section 4 investigates when and how to partition the problem; and Section 5 finds the minimal number of lots necessary to achieve the minimal makespan. Section 6 gives some simple examples. Section 7 introduces an exponential model with sub-lots, and Section 8 develops polynomial heuristics for the same purpose. Sections 9 and 10 take care formally of the issues of setups and integrality respectively. Section 11 introduces modifications that may be required for applying the model in practice. Finally, Section 12 concludes the paper with a brief list of related research questions.

2. The Formal Problem

[P] A batch of m items has to be processed sequentially on n machines, M_1, M_2, \dots, M_n . Each item requires T_i time units of processing on M_i ; for all i . Prior to processing the first unit, M_i requires a setup time of SU_i ; for all i . Transferring a lot of any size (up to and including m items) from M_i to M_{i+1} costs C_i , and takes TT_i time units; $i = 1, 2, \dots, n-1$. It is required to minimize the makespan subject to a budget constraint on the total transferring expenditure, B . ■

Definition: The symmetric problem is obtained from [P] by reversing the order of the machines. In this paper we use the term **symmetry** to refer to the relationship between [P] and the symmetric problem. ■

Setups in [P] become tear-downs in the symmetric problem; otherwise, the symmetry is perfect. The symmetric problem has the same minimal makespan as [P], and can be solved by the same lots--in reversed order. Symmetry is instrumental in proving most of the results below, starting with the next theorem:

Theorem 1: *Any feasible makespan can be realized in such a manner that both M_1 and M_n will process the whole batch continuously, although intermediary machines may have to idle intermittently.*

Proof: Trivial for M_1 , and by symmetry for M_n . ■■■

Borrowing PERT/CPM terminology, the theorem simply suggests adopting "early start" on M_1 and "late start" on M_n .

An item for our purpose may actually be a set of several units, say a dozen, if the policy is to

produce and transfer in dozens. For convenience, assume that at time 0 all the machines are free, but not yet set up for the batch. This assumption is not restrictive: if M_i is busy at time 0, say until time t , simply add t to SU_i . Similarly to [19], we also assume that transporting the lots is done independently of operating the machines. That is, the machines can continue working while the lots are being transferred. This assumption is appropriate in environments where dedicated resources are assigned to transferring items between stations. It is also appropriate if the transfer time, TT , is negligible. In addition, it may be possible for the operators of machines that idle between lots to handle the transfers. Other assumptions about this issue exist in the literature; e.g., see [6].

We use the budget constraint as an approximate way to allocate transportation resources to the various machines. In an environment where many such transfers are called for, and transportation is handled by a central department, this is equivalent to treating the transportation department as a profit center that sells transportation services to the jobs.

[17] includes an analytic model where a prespecified number of vehicles are available. The solution specifies lots that are large enough to make it possible for the vehicles to return in time for the next transfer. This solution can be easily implemented for adjacent pairs of machines in the present problem. Nevertheless, it is difficult to generalize this solution when the same vehicle can serve more than one pair.

The major effect of TT_i on the makespan is increasing it by a constant, namely ΣTT_i . This is true since we do not specify that the same vehicle has to handle all the transfers. Therefore, there is no need to wait for a vehicle to return from its former transfer before dispatching the current lot.

In addition, TT_i may influence the issue of whether machines downstream can be set up in time to process the first item that reaches them. (If they don't, a binding constraint is introduced.) Until Section 9 we assume $SU_i = 0$; for all i ; therefore, for determining the optimal lots, we can also assume $TT_i = 0$; for all i .

What is the potential for makespan reduction here? The minimal makespan can always be realized by transferring the items one-by-one. Therefore, the minimal makespan is the processing time on the slowest machine plus the processing time of *one item* on all the others. Now subtract this from $m\Sigma T_i$ to obtain the **maximal makespan reduction (MMR)**:

$$MMR = (m - 1)(\Sigma T_i - \text{Max}\{T_i\}). \quad (1)$$

In the example illustrated in Figures 1 through 4, the MMR is $(250 - 1)(1 + 2 + 1 + 3 - 3) = 996$. Therefore, the reduction in Figure 1, 375, is 37.7% of the MMR. Similarly, in Figures 2 through 4 the

reductions are 50.2%, 52.2% and 59.7% of the MMR respectively. The reduction in the example illustrated in Figure 5(b) is 100% of its MMR.

Similarly to [18], our stance in this paper is that as long as the makespan is minimized it is preferable to use as few transfers as possible. In this spirit we will show that $O(\log m)$ transfer lots will often suffice to achieve the maximal makespan reduction even if the budget is not binding.

We conclude this section with a definition of partitions.

Definition: A set of machines that are required to work continuously is called a **partition set**, or simply a partition. We stipulate that a partition set must include M_1 and M_n (as per Theorem 1). A partition can be read as a list of machines, or as a list of pairs. For instance, if M_i and M_j are adjacent to each other in a partition we say the partition includes the pair (i, j) . For convenience, we list a partition by only listing the indices of the machines in it. When a partition includes r machines in addition to M_1 and M_n , we may list it as $\{p(0)=1, p(1), p(2), \dots, p(r), p(r+1)=n\}$, or $\{p(s)\}_{s=0, r+1}$. ■

3. A Preliminary Model

In this section we start treating a **relaxed problem**, where fractional items can be transferred. The relaxed problem is solved by the **relaxed solution**, as opposed to the **integral problem/solution**. To avoid an excessive gap between the relaxed solution and the integral one, we stipulate that in a relaxed solution all lots should be ≥ 1 . This restriction is also instrumental for demonstrating that the number of lots required to achieve the maximal possible makespan reduction is often $O(\log m)$. We refer to instances where lots are allowed to be < 1 as **super-relaxed**.

Using Theorem 1, we specify that M_1 and M_n should operate continuously, while the intermediate machines may idle between lots. We use our first operating procedure, i.e., the lots retain their composition across all machines unless a partition is involved. We denote the size of lot j by L_j ; $j = 1, \dots, k$; we may also use L_j informally as the name of lot j . We denote the cumulative sum of the first j lots by S_j ; e.g., $S_1 = L_1$, and $S_k = m$. Under partition, L_j may be different for each subset, so formally we should use a double index to identify L_j and S_j . Nevertheless, it is possible to use the simpler notation without causing confusion. Our formal problem is now:

[Pk] Solve [P] under the following assumptions: (i) $TT_i = SU_i = 0$, for all i ; (ii) the lots retain their composition across all machines in each subset; (iii) fractional items may be transferred, but $L_j \geq 1$; for all j . ■

Definition: A solution is called **well-behaved** if (i) M_1 and M_n operate continuously as per Theorem 1; and (ii) M_i ($i = 2, \dots, n-1$) can process each lot as soon as it becomes available from M_{i-1} (i.e., lots are neither delayed at M_{i-1} nor queued at M_i). If a lot can reach a machine before the machine is ready for it, we say there is a **conflict**. ■

Figures 3 and 4 illustrate well-behaved solutions. Figures 1 and 2 illustrate conflicts; e.g., in both cases the second lot from M_1 can reach M_2 at time 250, but M_2 is not ready for it until time 375.

It turns out that if fractional items may be transferred, the optimal solution tends to be well-behaved. The only exception may occur due to the restriction $L_j \geq 1$, which constitutes an integrality constraint when it is binding. Thus, the optimal super-relaxed solution is well-behaved.

This is true because if conflicts exist, lots which have to wait could have been increased without increasing the makespan. Since the sum of all the lots is constant (m), some preceding lot could have been decreased, thus feeding M_n sooner, and reducing the makespan. This argument fails if by reducing the preceding lot we violate the constraint $L_j \geq 1$ for it.

We also assume in this section that partition will be neither specified nor required; i.e., there exists a well-behaved solution that can be found without specifying any partition. Without partition, under our operating procedure the maximal number of transfers allowed by the budget is $k = \text{INT}(B/\Sigma C_i)$. To obtain a well-behaved solution we must have

$$L_{j+1}(T_1 + T_2 + \dots + T_{n-1}) = L_j(T_2 + T_3 + \dots + T_n) ; j = 1, 2, \dots, k-1. \quad (2)$$

That is, the time it takes to process L_{j+1} on the first $n-1$ machines should equal the time it takes to process L_j on the last $n-1$ machines. Thus L_{j+1} will reach M_n exactly when it is ready for it, as it should in a well-behaved solution. Figure 6 illustrates this point: $L_1 = 100$ items starts processing on M_2 at time 100 and finishes on M_4 at time 700, a total of $100(1+2+3) = 600$ time units; $L_2 = 150$ units starts processing on M_1 at time 100 also, and finishes on M_3 at time 700--just when M_4 can start processing it--also a total of $150(1+1+2) = 600$ time units.

Insert Figure 6 about here

Define

$$Q_{i,j} = (T_{i+1} + T_{i+2} + \dots + T_j) / (T_i + T_{i+1} + \dots + T_{j-1}),$$

and let $Q = Q_{1,n}$; then the set (3) replaces the set (2)

$$L_j = L_{j-1}Q = L_1Q^{j-1} \quad ; \text{ for all } j \geq 2. \quad (3)$$

In addition we have

$$L_1 + L_2 + \dots + L_k = m. \quad (4)$$

In order to satisfy (4), we use the familiar geometric progression to obtain

$$L_1 = \begin{cases} m(1 - Q)/(1 - Q^k) & ; \quad Q \neq 1 \\ m/k & ; \quad Q = 1. \end{cases} \quad (5)$$

Along with (3), this suffices to specify all the lots. To continue, let us investigate the conditions under which (3) and (5) yield a well-behaved solution. The question is whether there will be conflicts at the intermediary machines. Checking the following $n - 2$ **feasibility conditions** gives the answer.

$$Q_{1,n} \geq Q_{1,p} ; 2 \leq p \leq n-1. \quad (6)$$

The conditions are equivalent to comparing the lot sizes obtained in the solution to those obtained by stipulating that M_p will work continuously, *given* L_1 . Under this stipulation (3) determines L_2, L_3 , etc., but with $Q = Q_{1,p}$ instead of $Q = Q_{1,n}$. As long as the lots are not smaller than those required by M_p to operate continuously, they cannot reach M_p before it is ready for them. Note that the conditions are invariant under the number of lots, the size of the first lot, and m . In the example illustrated in Figure 6, $Q_{1,4} = Q_{1,3} = 1.5$. That is, one of the feasibility conditions is satisfied as an equality; therefore M_3 operates continuously even though no partition is enforced.

If conflicts do occur in intermediary machines, we can specify that the machines involved shall operate continuously, as do M_1 and M_n . That is, we partition the problem (see Section 4).

Let $[(i)]$ denote the numerical value obtained by applying (i). If $[(5)] < 1$ or $[(5)]Q^{k-1} < 1$, we use the following procedure:

The Adjustment Procedure:

(a) $Q \geq 1$ AND $[(5)] < 1$:

1. Set $L_1 = S_1 = 1$, and let $i = 1$ (recall $S_i = \sum_{j=1,i} L_j$);
2. if $S_i < m$, let $L_{i+1} = L_i Q$ (as per (3)), $S_{i+1} = \min\{S_i + L_{i+1}, m\}$, $i = i + 1$; proceed to the next step;
3. if $S_i = m$, set $K = i$, $L_K = m - S_{K-1}$, and proceed to the next step; otherwise, return to the former step.
4. iterating backwards from $j = K$, if $L_j < 1$ increase L_j to 1 by decreasing L_{j-1} ; upon encountering j such that $L_j \geq 1$, STOP.

(b) $Q < 1$ AND $[(5)]Q^{k-1} < 1$:

Apply Part (a) of the procedure to the symmetric problem. ■

If $Q \geq 1$, the lots under (3) (except possibly the last one when $L_1 \neq [(5)]$) are non-decreasing, so if $L_1 = [(5)] \geq 1$ all the lots are ≥ 1 , and there is no need for adjustment. A symmetric observation holds if $Q < 1$ and $L_1 = [(5)]Q^{k-1} \geq 1$. Otherwise, it is easy to verify that the procedure ensures $L_j \geq 1$; for all j . Let * superscripts denote optimal values, then

Theorem 2: *If the feasibility conditions are satisfied, $[(5)] \geq 1$ AND $[(5)]Q^{k-1} \geq 1$, then $L_1^* = [(5)]$; else, if the feasibility conditions are satisfied, then The Adjustment Procedure yields an optimal solution. (See Appendix for proof.) ■■■*

Let MS^r denote the relaxed makespan, and MS^* denote the optimal integral solution, then $MS^* - MS^r = h^* \geq 0$ is the difference in makespan due to the integrality constraints. We conclude this section by developing a simple upper bound for h^* , still under the assumption that no partition is involved, and that $SU_i = 0$; for all i . Under partition, the bound will apply to each part separately. The sum of the bounds of the parts will then be the bound we seek. To continue, let f_i be the fractional part of S_i , and define

$$e_i = \begin{cases} 1 - f_i & ; f_i > 0 \\ 0 & ; f_i = 0. \end{cases}$$

Theorem 3: $h^* \leq \min\{\text{Max}_i\{e_i\}_{\sum_{j=1,n-1} T_j}, \text{Max}\{f_i\}_{\sum_{j=2,n} T_j}\}$
 $< \sum_{j=1,n} T_j - \text{Max}_i\{T_1, T_n\}.$

Proof: We introduce two heuristics:

Heuristic 1: Round all the S_i values up to the nearest integer, i.e., $S_i + e_i$. That is,
the new value for L_i will be $(S_i + e_i) - (S_{i-1} + e_{i-1})$.

Heuristic 2: Truncate all the S_i values down to the nearest integer, i.e., $S_i - f_i$.

See Appendix for proof that Heuristic 1 increases the makespan of the relaxed solution by $\text{Max}\{e_i\}_{\sum_{j=1,n-1} T_j}$, Heuristic 2 by $\text{Max}\{f_i\}_{\sum_{j=2,n} T_j}$, and both yield feasible solutions. ■■■

Users who are satisfied with approximate solutions can use one of these heuristics instead of applying the results of Section 10.

4. Partitioning the Problem

The simplest way to enforce the feasibility conditions (6), when some of them are violated, is to look for the largest index p which maximizes $Q_{1,p}$, and specify the partition $\{1, p, n\}$. This creates $n - p - 1$ new feasibility conditions downstream from M_p , i.e.,

$$Q_{p,n} \geq Q_{p,j} ; p+1 \leq j \leq n-1,$$

and more machines may have to be added to the partition set between p and n . In this section we investigate this and other partitions of the problem.

Definition: A pair (i, j) is called **feasible** if $Q_{i,j} \geq Q_{i,p}$; for all $i \leq p \leq j$. That is, M_i can feed M_j as per (3) and (5), with both of them operating continuously. ■

Note that adjacent pairs, e.g., $(i, i+1)$, are always feasible.

Definition: A partition is called **feasible** if all its pairs are feasible. ■

Specifying any feasible partition and using (3) and (5) for each part yields a well-behaved solution.

Definition: The partition implied by the feasibility conditions is called the **minimal partition**. That is, if $\{p(s)\}_{s=0,r+1}$ denotes the minimal partition, then

$Q_{p(s),p(s+1)} = \text{Max}_{j > p(s)} \{Q_{p(s),j}\}$; for all $0 \leq s \leq r$, and $p(s+1)$ is the largest index j that maximizes $Q_{p(s),j}$. We refer to the minimal partition as **MINPARTIT**, and note that it is feasible by construction. ■

Definition: The (feasible) partition which includes all the machines is the **maximal partition**, denoted by **MAXPARTIT**. ■

Definition: The **medium partition**, is the feasible partition with the maximal number of machines in it, such that given a large enough budget it will be possible to realize MMR, as per (1). We refer to it as **MEDPARTIT**. ■

It may happen that specifying **MEDPARTIT** will cause the makespan to increase relative to **MINPARTIT**. The definition merely says that *given a large enough budget* MMR can be achieved. The first example in Section 6 illustrates this point.

MAXPARTIT contains all other partitions, feasible or infeasible. **MINPARTIT**, as the name suggests, is contained in any feasible partition (see Theorem 6). And, any feasible partition properly contained in **MEDPARTIT** enables MMR. We show how to construct **MEDPARTIT** and substantiate these claims below.

To continue our investigation of partitions, we present a few theorems. The proofs are deferred to the Appendix. Theorem 4 is the motivation for the definition of **MEDPARTIT**. It indicates that

partitions may limit the maximal possible gain. This is not surprising, since they impose additional constraints on the problem. Before stating the theorem we define $MMR_{i,j}$ as the maximal makespan reduction for pair (i, j) . As in (1), we obtain

$$MMR_{i,j} = (m - 1)(\sum_{s=i,j} T_s - \text{Max}\{T_s\}_{s=i,j}). \quad (7)$$

Theorem 4: For any partition $\{p(0)=1, p(1), p(2), \dots, p(r), p(r+1)=n\}$ (feasible or not), $MMR_{1,n} \geq \sum_{i=0,r} MMR_{p(i),p(i+1)}$. ■■■

It is easy to construct examples where the inequality is strict (see Figure 7). We also want to show that the minimal partition does not imply any potential losses in MMR. Thus, for the minimal partition Theorem 4 is satisfied as an equality. We state this result below as Theorem 5. But first, we present a lemma.

Insert Figure 7 about here

Lemma 1: Let M_i, M_p , and M_j be any set of three machines such that $i < p < j$ and $T_p > \text{Max}\{T_i, T_j\}$, then (i, j) is an infeasible pair. ■■■

A direct corollary of Lemma 1 is that at least one of the slowest machines must belong to any feasible partition, including MINPARTIT and MEDPARTIT.

Theorem 5: Let $\text{MINPARTIT} = \{p(0)=1, p(1), p(2), \dots, p(r), p(r+1)=n\}$, then

$$MMR_{1,n} = \sum_{i=0,r} MMR_{p(i),p(i+1)}. \quad \blacksquare$$

Theorem 6: Let PARTIT be any feasible partition which is not identical to MINPARTIT, then MINPARTIT must be properly contained in PARTIT. ■■■

We also need another lemma, which specifies sufficient (but not necessary) conditions for a pair to be feasible.

Lemma 2: Let (i, j) be a pair such that $T_k \leq \min\{T_i, T_j\}$; for all $i < k < j$, then (i, j) is a feasible pair. ■■■

We are now ready to identify MEDPARTIT. The procedure is based on the results above, and observation of the proof of Theorem 4 (see Appendix). We are looking for the largest set of indices, $\{s\}$, such that (i) T_s is monotone non-decreasing for any s which is smaller than the largest index of a slowest machine; and (ii) T_s is monotone non-increasing for any s which is larger than the smallest index of a slowest machine.

In more detail, by Theorem 5 we can start with the minimal partition, and add machines between its pairs where possible. It is enough to concentrate on one pair at a time, say $(p(s), p(s+1))$. There are up to three types of pairs in **MINPARTIT**: (i) pairs where $T_{p(s)} < T_{p(s+1)}$; (ii) pairs where $T_{p(s)} = T_{p(s+1)}$, which can only happen if both of these values are maxima; and (iii) pairs where $T_{p(s)} > T_{p(s+1)}$. If we have a pair of the first or second type, we look for the smallest index i such that $p(s) < i < p(s+1)$ and $T_i \geq T_{p(s)}$, and include it in **MEDPARTIT**. By Lemma 2, $(p(s), i)$ is a feasible pair. $T_i \leq T_{p(s+1)}$ (otherwise, by Lemma 1, the pair $(p(s), p(s+1))$ would not be feasible, and it must be feasible because it belongs to **MINPARTIT**), so $(i, p(s+1))$ is also a pair of the first or second type (though not necessarily feasible). Thus we can repeat the procedure iteratively for the pair $(i, p(s+1))$, where i takes the place of $p(s)$ above. When no such i exists, we move to the next pair of **MINPARTIT**. Pairs of the third type are treated symmetrically to pairs of the first type.

Any feasible partition contained in **MEDPARTIT** also has the same structure, i.e., (i) T_s is monotone non-decreasing for any s which is smaller than the largest index of a slowest machine; and (ii) T_s is monotone non-increasing for any s which is larger than the smallest index of a slowest machine. Therefore, such partitions also enable MMR.

Makespan Reduction as a Function of the Number of Transfers

Let (i, j) be a feasible pair, specified as part of a partition, and let $MR_{i,j}(k)$ denote the marginal makespan reduction associated with allowing k transfer lots relative to $k-1$ lots for the subset M_i, \dots, M_j . Let $K_{i,j}$ be the number of lots required to achieve $MMR_{i,j}$ (as indicated by running The Adjustment Procedure for the subset, or see (11) and (12) below), then

$MR_{i,j}(1) = 0$ (this transfer is essential),

$$m(\sum_{s=i,j-1} T_s) Q_{i,j}^{k-1} (1-Q_{i,j})^2 / [(1-Q_{i,j}^{k-1})(1-Q_{i,j}^k)] \quad ; Q_{i,j} \neq 1, 1 < k < K_{i,j}$$

$$MR_{i,j}(k) = \begin{cases} m(\sum_{s=i,j-1} T_s) / (k(k-1)) & ; Q_{i,j} = 1, 1 < k < K_{i,j}, \end{cases} \quad (8)$$

$$MR_{i,j}(K_{i,j}) = MMR_{i,j} - MR_{i,j}(K_{i,j} - 1). \quad (9)$$

(8) is derived from (A9) (see Appendix). Next, we have a convexity property:

Theorem 7: *The series $\{MR_{i,j}(k)\}_{k=2,3,\dots,m}$ is monotone decreasing. ■■■*

Furthermore, the first extra transfer can yield at least half of the maximal makespan reduction, $MMR_{i,j}$, i.e.,

Theorem 8: $MR_{i,j}(2) > MMR_{i,j}/2$. ■■■

In case of partition, Theorem 8 applies to each part separately, but not necessarily to the whole set. For instance, in the example illustrated in Figure 7(a), where **MAXPARTIT** is specified, the total makespan reduction is 18--less than half of $MMR = 99$. Nevertheless, it is easy to verify that under **MEDPARTIT** or any feasible partition contained in it (e.g., **MINPARTIT**), Theorem 8 does apply to the whole (Figure 4).

Identifying the Optimal Partition

Given a feasible partition (e.g., **MINPARTIT**, **MEDPARTIT**, or **MAXPARTIT**), Theorem 7 suggests allocating the budget to the parts of the problem by a heuristic procedure, to which we refer as **The Greedy Heuristic**. First, we list and sort all the $MR_{i,j}(k)/\sum_{s=i,j-1} C_s$ values. Then, we specify as many transfers as possible from the top of the list. That is, we use transfers that yield the highest marginal gain per dollar--as long as the budget allows. This also makes it possible to find the most economical budget, if the budget is not the result of real technical constraints. To that end, we simply add transfers from the top of the list as long as the extra expense is justified by the marginal contribution. If the heuristic allocates the budget completely, the solution is optimal. Otherwise, it may happen that by giving up some of the transfers indicated by the heuristic, larger makespan reductions may be obtained in other pairs. This is the heuristic presented in [17] for **MAXPARTIT**. The main problem with The Greedy Heuristic is that we need to know the partition in advance, or check all the possible candidate partitions (which would lead to an algorithm exponential in n).

It is possible, however, to solve this problem to optimality by dynamic programming (DP). The Appendix presents a DP algorithm that finds the optimal partition and allocation in $O(n^3m^2)$ or $O(n^3m \log m)$, depending respectively on whether or not $Q_{i,j}$ is likely to be 1.

5. Achieving the Maximal Gain with the Minimal Number of Lots

Let $MS_{i,j}$ be the minimal makespan possible for a feasible pair (i, j) when k transfers are allowed. $MS_{i,j}$ is measured from the start of the batch on M_i to its finish on M_j . Observe that we have to process the first lot on the first $j - i$ machines and then process the whole batch on M_j . This leads to the following bound

$$MS_{i,j} \geq L_1(T_i + T_{i+1} + \dots + T_{j-1}) + mT_j \quad (\geq (T_i + T_{i+1} + \dots + T_{j-1}) + mT_j). \quad (10)$$

The second inequality arises because $L_1 \geq 1$. By symmetry we also have

$$MS_{i,j} \geq L_k(T_{i+1} + T_{i+2} + \dots + T_j) + mT_i \quad (\geq (T_{i+1} + T_{i+2} + \dots + T_j) + mT_i).$$

Either or both of these inequalities can be strict, but under (3) and (5) they are both satisfied as equalities. For that to happen, both M_i and M_j have to operate continuously. Our objective in this section is to find the minimal number of transfers required to minimize $MS_{i,j}$. We denote this number by $K_{i,j}$ (corresponding to K in The Adjustment Procedure). This number may be slightly impacted by the integrality constraints that we relaxed for the present, but it gives a good approximation for the integral version as well. Our procedure for finding $K_{i,j}$ is simple. First, if $Q_{i,j} > 1$, then (3) leads to increasing lots, and L_1 is the smallest lot. Since $L_1 \geq 1$, by specifying $L_1 = 1$ and using (3) for the other L_j values we obtain an increasing geometric series. (10) assures us that this policy will minimize the makespan, since it will hold as an equality. How many members should the series have to exceed m for the first time? The answer is:

$$K_{i,j} = \text{SUPINT}(\log[(Q_{i,j} - 1)m + 1]/\log Q_{i,j}) \quad ; \quad Q_{i,j} > 1, \quad (11)$$

where $\text{SUPINT}(x) = \text{smallest integer } \geq x$. Next, if $Q_{i,j} < 1$, then by symmetry

$$K_{i,j} = \text{SUPINT}(\log[(1/Q_{i,j} - 1)m + 1]/\log[1/Q_{i,j}]) \quad ; \quad Q_{i,j} < 1. \quad (12)$$

Finally if $Q_{i,j} = 1$, then $K_{i,j} = m$.

The procedure of calculating L_j by (3) based on $L_1 = 1$ is similar to our approach in The Adjustment Procedure. Note that as a result, we obtain

$$L_{K_{i,j}} = m - S_{K_{i,j}-1} \leq L_{K_{i,j}-1} Q_{i,j}.$$

The optimal solution is not necessarily unique here, since we have some degree of freedom in changing the lots following L_1 without violating (2). Nevertheless, the lots specified above are optimal. We say this not only in the sense that the makespan is minimized but also in the sense that only necessary transfers are specified. Similarly, if the bound is on the last lot instead of the first, we use the same procedure, but backwards. Here we use (12) below instead of (3), starting with $L_{K_{i,j}}$.

$$L_{j-1} = \begin{cases} L_j(1/Q) & ; \quad 3 \leq j \leq K_{i,j} \\ m - \sum_{j=2, K_{i,j}} L_j & ; \quad j = 2. \end{cases}$$

By observation of (11) or (12), if $Q \neq 1$, $K_{i,j}$ is $O(\log m)$. This is obviously good news to management. It also has a beneficial effect on the complexity of the procedures we employ.

6. Examples

To illustrate the first operating procedure, along with its shortcomings, let us look at some simple examples. In Examples 1 and 2, with four machines each, we look for the minimal makespan with two transfer lots. Examples 3 and 4, with three machines each, illustrate the budget allocation and its effect on the makespan.

Example 1: Let $m = 250$, $T_1 = T_2 = 1$, $T_3 = 2$, $T_4 = 3$, and assume the budget suffices for exactly two transfer lots. (We ignore the issue of optimal budget allocation here.)

Solution (see Figure 6): $Q_{1,4} = (T_4 + T_3 + T_2)/(T_3 + T_2 + T_1) = (3 + 2 + 1)/(2 + 1 + 1) = 1.5$, $Q_{1,3} = (T_3 + T_2)/(T_2 + T_1) = (2 + 1)/(1 + 1) = 1.5 \leq 1.5$, and $Q_{1,2} = 1 \leq 1.5$, \Rightarrow no partition is required. That is, $\text{MINPARTIT} = \{1, 4\}$; $\text{MEDPARTIT} = \text{MAXPARTIT} = \{1, 2, 3, 4\}$. Since $Q_{1,3} = Q_{1,4}$, M_3 will operate continuously under MINPARTIT , even though it is not specified explicitly in the partition. Solving for L_1 we obtain 100 units, and the total makespan is 1150. ■■

This example serves to illustrate an important point about MEDPARTIT : given a *large enough budget*, MEDPARTIT enables the maximal possible gain; but if we restrict the budget, as implied in this example, then partitioning at a machine which belongs to MEDPARTIT can increase the makespan. In this example, there would be an increase of about 8 time units associated with adding M_2 to the partition. We leave the computational details to the interested readers.

Insert Figure 8 about here

Example 2: As before, but interchange the order of M_2 and M_3 , i.e., $T_2 = 2$ and $T_3 = 1$. Here, if we try to stick to the former solution ($L_1 = 100$), there will be a conflict at M_2 (see Figure 8), since M_2 still processes L_1 when L_2 reaches it. Indeed, $Q_{1,2} = 2 > 1.5 = Q_{1,4}$. The total makespan is 1200. By specifying $\text{MINPARTIT} = \{1, 2, 4\}$ in this example--we can reduce the makespan to 1155 (see Figure 4), most of the way back to 1150. Note that this requires changing the size of L_1 from 83 between M_1 and M_2 to 107 downstream. Figure 3 illustrates the makespan for the same example under MAXPARTIT , again with a variable size for L_1 . The additional delay is 75 units, or about 6%. (For this example, the integrality constraints were satisfied by simple rounding, and therefore the makespan may be slightly off.) ■■

Example 3 (see Figure 7): Let $m = 10$, $T_1 = T_3 = 10$, $T_2 = 1$, $C_1 = C_2 = 1$, and $B = 20$.

Solution Under First Operating Procedure: Each compound transfer costs $C_1 + C_2 = 2$, so we can have up to 10 lots. $Q = (1 + 10)/(10 + 1) = 1$. $\Rightarrow L_1 = 1$, and we obtain 10 equal lots of one item each.

Checking the feasibility conditions (6), $1 > 0.1$, hence no conflicts will occur. The makespan is 111. ■■

Solution Subject to All Machines Operating Continuously: This is a forced partition, namely MAXPARTIT. Applying (12) to pair (1, 2), we obtain $\text{SUPINT}(\log(10 - 1)10 + 1]/\log 10) = \text{SUPINT}(\log 91/\log 10) = 2$. The same numeric result applies to (2, 3), due to symmetry. Hence, the budget is not binding. By (5) we obtain L_1 for pair (1, 2) = 9/.99, but since this leads to $L_2 < 1$, The Adjustment Procedure yields $L_2 = 1$, and $L_1 = 9$. For the second pair, by symmetry, $L_1 = 1$, and $L_2 = 9$. The total makespan is 192 (and not 191), since M_2 is constrained to work continuously, and hence it cannot start until time 91. M_2 feeds M_3 for the first time at time 92. M_3 , though it can start immediately, requires 100 time units. In this example Theorem 4 is satisfied as a strict inequality. Indeed note that M_2 does not belong to MEDPARTIT. ■■

Example 4: Let $m = 5$, $T_1 = T_2 = 1$, $T_3 = 5$, $C_1 = 1$, $C_2 = 10$, and $B = 23$.

Solution Under First Operating Procedure (see Figure 5[a]): $Q = (1 + 5)/(1 + 1) = 3$. $C_1 + C_2 = 11$, so the budget suffices for two composite transfers, leaving \$1 unused. By (5) $L_1 = 1.25$, and the completion time is 27.5. Now, if we consider the integrality constraints, we are indifferent between $L_1 = 1$ or $L_1 = 2$. In the former case the second lot of 4 items will get to M_3 at time 9, and it will still require $5 \cdot 4 = 20$ time units to finish the batch. In the latter case, M_3 receives the first lot at time 4, and finishes at time 29 again. ■■

Solution Subject to the Maximal Partition: This is a forced partition again, but note that this time MEDPARTIT = MAXPARTIT. $Q_{1,2} = 1$, so we can use up to 5 lots there. Applying (11) to pair (2, 3), we obtain $\text{SUPINT}(\log(5 - 1)5 + 1]/\log 5) = \text{SUPINT}(\log 21/\log 5) = 2$. That is, there is room for up to one extra transfer there. Here our budget is binding. By (8) we get $MR_{2,3}(2) = 4$; $MR_{1,2}(2) = 2.5$; $MR_{1,2}(3) = 0.8333$; $MR_{1,2}(4) = 0.4167$; $MR_{1,2}(5) = 0.25$. Dividing these values by the respective transfer costs, the contribution per dollar of $MR_{2,3}(2)$ is 0.4, while the other values remain unchanged. Therefore, the greedy heuristic will specify three extra transfers between 1 and 2. Then, it will try to introduce an extra transfer for (2, 3), but the remaining budget of \$9 will be \$1 short. In this example an exact algorithm (such as the DP model) can yield the optimal allocation, namely, one extra transfer for (2, 3) and two for (1, 2). The resulting makespan is 27.667. The best integral solution here yields 28. (Note $27.667 > 27.5$ but $28 < 29$. Since $27.667 > 27.5$, the dynamic programming algorithm would not specify a partition, even though with the integrality constraints it is better.) ■■

Optimal Solution (See Figure 5[b]): Allocate \$3 to three lots from M_1 , and \$20 to two transfers from M_2 , thus utilizing the budget fully. Let M_1 send the first item as the first lot, and then two lots of two

items each. M_2 will be able to send the first item to M_3 at time 2, and start the next four items at time 3 (M_2 will idle between 2 and 3). At time 7 M_2 and M_3 both finish their current load, and M_2 sends the four items to M_3 "just in time." Thus $MS_{1,3} = 27$. ■■

The last solution happens to comply with the integrality constraints and achieves the maximal makespan reduction as per (1). Therefore, it is the optimal solution. Note that this integral solution gives a shorter makespan than the 27.5 we got before with the relaxation. It compares even more favorably with the integral solution's 29. Thus, our first operating procedure leaves something to be desired. Indeed it is intuitively clear that when the transfer costs and the potential marginal gain differ from stage to stage, forcing the model to specify the same number of transfers across all stages may be wasteful. This motivates our second operating procedure.

7. Introducing Sub-lots into the Solution

In this section we treat [Pk] without the constraints $L_j \geq 1$; leaving them in would complicate the presentation considerably. Thus, we look for the optimal super-relaxed solution. Section 10 develops integral solutions for this section's operating procedure (as well as for the former operating procedure).

Examine the optimal solution of Example 4. At first, the lot size remains constant for all machines (one unit). Then, two lots from M_1 to M_2 are combined to one lot from M_2 to M_3 . Note that though M_2 has to idle after the first lot, it does so only after having dispatched all the items it processed. One way to describe the optimal solution of Example 4 is to say that the batch is divided to two parent lots. The first is processed on all machines "as is," while the second is further divided to two sub-lots on M_1 . Each machine processes each parent lot continuously. The intermediary machine idles between parent lots. Note that the solution is well-behaved.

This leads us to an operating procedure where we retain the same parent lots across all machines, but allow sub-lots within each parent lot. Machines can idle between parent lots, but not between sub-lots. Since the number of sub-lots in parent lot i and parent lot j need not be equal. Therefore the budget should not necessarily be allocated to the parent lots equally. Thus we are presented with a sub-problem of allocating the budget to the parent lots. We need to solve the sub-problem before allocating the sub-budget within the parent lot. Initially, we assume all intermediary machines idle between parent lots. By Theorem 1 we know that M_1 and M_n do not have to idle. Later we will discuss partitions, and thus take care of conflicts in intermediary machines.

Assume for a while that we know how many parent lots there are, and to how many sub-lots each parent lot is divided on each machine. We refer to this information as **the grouping information**. We proceed to deduce the optimal lot (and sub-lot) sizes from the grouping information.

The size of parent lot i , still denoted by L_i , is constant across all the machines. The sub-lots may be different for different machines, even if their number is the same. Observe that under our assumptions each parent lot is processed as a batch under **MAXPARTIT**. When a parent lot is not divided to sub-lots on some machine we may refer to it as a sub-lot of itself. Let:

- $Y_{i,j,k}$ be the relative size of sub-lot j , emanating from M_k , and belonging to L_i . ($\sum_j Y_{i,j,k} = 1$.)
- $ST_{i,k}$ be the time the first sub-lot of L_i starts processing on M_k .

Then, since M_1 and M_n operate continuously, the first sub-lot of L_i starts processing on M_1 at time

$$ST_{i,1} = T_1 \sum_{p=1, i-1} L_p, \quad (13)$$

and on M_n at time

$$ST_{i,n} = L_1 \sum_{k=1, n-1} Y_{1,1,k} T_k + T_n \sum_{p=1, i-1} L_p. \quad (14)$$

Let L_i^t denote a tentative value for L_i , then to calculate all the L_i and $Y_{i,j,k}$ values we start with $L_1^t = 1$ and proceed recursively, as outlined below.

- (1) At stage i , given L_i^t and $Y_{p,j,k}$ for $p < i$ and all j, k , calculate $Y_{i,j,k}$ for all j, k ; if $i = K$, go to (3); else, set $i = i + 1$ and go to (2);
- (2) Given L_p^t and $Y_{p,j,k}$ for $p < i$ and all j, k , calculate L_i^t and return to (1)
- (3) For $i = 1$ to K let $L_i = mL_i^t / \sum L_p^t$

The details of the calculations are deferred to the Appendix. By observing these details we can see that unless there are conflicts between parent lots on intermediary machines we obtain well-behaved solutions.

We still operate under three assumptions: (i) we have the grouping information; (ii) all intermediary machines are allowed to idle intermittently; and, (iii) there are no conflicts at intermediary machines. Given these assumptions, we have an efficient procedure to calculate all the lot and sub-lot sizes. It is when lifting the assumptions that our method may become exponential.

For instance, to lift the first two assumptions we can generate all the possible groupings and partitions which the budget allows. It is easy to see that this can be done in exponential time. We then solve for each such grouping/partition as described above.

As for conflicts at intermediary machines, we check for them by the following set of conditions:

$$ST_{i,k} \geq ST_{i-1,k} + L_{i-1}T_k \quad ; k = 2,..., n-1, \text{ all } i. \quad (15)$$

Where we calculate the ST values recursively (using (13) for $k = 2$), i.e.,

$$ST_{i,k} = ST_{i,k-1} + L_i Y_{i,1,k-1} T_{k-1} \quad ; k = 2,..., n-1.$$

If the conditions hold, the solution is a valid candidate. Otherwise, that particular grouping/partition is rejected. The best candidate solution is the method's final output.

8. Polynomial Heuristics with Sub-lots

In this section we introduce simple rules based on results from the former analysis, to obtain polynomial heuristics.

For a while, let us assume that no partition is required. The technique we develop under this assumption can then be applied to the separate parts of the problem, if we partition it later. Furthermore, it will indicate one possible "quick and dirty" method of partition.

Under this assumption, we have to allocate the budget to parent lots, and--within each parent lot--to the different machines. This gives us the grouping information. Given the grouping information, Section 7 lists all the necessary calculations.

The basic rule we use for allocating the budget among the parent lots is to make the allocation as equal as possible. To the extent it is not possible to allocate the budget evenly, we make the allocation monotone non-decreasing, or monotone non-increasing. To choose between those two, we have the following rule of thumb: if the first machine is faster than the last, choose the non-decreasing order; otherwise, choose the non-increasing order. (This rule works well for Example 2. The rationale behind it is that it allocates more to the larger parent lots.)

Allocating the budget evenly does not yet tell us how many parent lots we should use. Dividing the budget by $\sum C_i$ will give us an upper bound for that purpose. Another plausible (heuristic) upper bound may be obtained by using $K_{i,j}$, as per (11) or (12). Next we may check for lower numbers of parent lots. We continue considering less and less parent lots until one of the following happens: (i) we get down to a single parent lot; (ii) our results start to deteriorate instead of improving. Symmetrically, we can start with one parent lot, and then increase the number.

At this stage, it remains to allocate the budget within the parent lot. To do that we propose to use the solution of the maximal partition procedure as applied to the parent lot (instead of to the whole batch). Note that we do not really have to know m , or L_p , to use this solution. Hence we have enough data to apply it. Also recall that the partition is known (i.e., **MAXPARTIT**), and The Greedy Heuristic as well as the DP solution can supply the necessary allocation.

Theoretically, by applying the maximal partition procedure we minimize the makespan of the parent lot, s.t. no idling between sub-lots. In fact, what we wish to do is to maximize the size of the particular parent lot, without exceeding the time frame allotted to it. That is, we wish to maximize the throughput. Minimizing the makespan for a known number of items, however, is dual to maximizing the throughput during a set period. Therefore, this solution is appropriate here.

Having made the allocation to parent lots, we now use (15) to determine whether there are conflicts. If so, we can partition and solve the problem in the following way:

- (i) take the machine which is most conflicted (that is, has conflicts for the largest percentage of time), and add it to the partition set.
- (ii) allocate the budget to the parts exactly as was the allocation to the machines belonging to each part before. Then, solve the problem for each sub-problem separately.

Alternately, when we check a new partition we may simply scrap it if there are any conflicts. In particular, if conflicts occur when we try to solve without partition, this would imply that we must partition the problem, one way or another. To do that, we can use some of the following ways:

- Partition at machines which conflict while applying the heuristic. (These will probably be slow machines.)
- Try **MINPARTIT** or **MEDPARTIT**
- If there are few machines, try all possible partitions.

To allocate the budget to the parts, we have two alternatives: (i) if we choose to determine the partition by the first method, we can allocate the budget accordingly, as discussed above; (ii) in all cases, including the former, we can also use dynamic programming similarly to the procedure above. Of course, the latter will be more time-consuming.

Note that if we partition by dynamic programming, allocating the budget to the parts can be done together with the partitioning. Otherwise, we start with a given partition, and optimize the allocation of the budget to the parts. To do that observe that the savings in the various parts are additive, so a knapsack DP algorithm can be used. For this purpose, we have to build table functions for each part

showing how much can be saved within it for any given budget allocation. We do this by allocating the budget as evenly as possible *within each part*. If we get conflicts within a part under a certain number of parent lots, we look for a different number of parent lots.

The complexity of the heuristic depends on the choices we make. For instance, if we choose to check out all the possible partitions, we have exponential complexity. All the other choices yield polynomial complexities.

9. Dealing with Setups

(16), below, is a set of conditions which ensure that the setups cannot become critical. If (16) is violated for some k , then SU_k is *potentially* critical.

$$SU_k \leq \text{Max}_p \{SU_{k-p} + \sum_{j=i-p, i-1} (T_j + TT_j)\}; \text{ all } 2 \leq k \leq n. \quad (16)$$

For every potentially critical setup we have to check the solution for conflicts. If there are conflicts, the setups involved are critical indeed. We now list several treatments we can specify if some setups are critical.

The simplest solution is to delay the whole schedule by $\max_i \{D_i\}$, where D_i is the amount by which SU_i conflicts with the original schedule. That is, the most critical setup determines the delay. If we do this, however, it may become possible to specify less transfers upstream of the most critical setup, and more setups downstream. This may recapture some of the delay within the same budget.

To minimize the makespan without a budget constraint, we can always partition the problem at the most critical machine, and look for a solution which starts feeding it as soon as the setup is completed. This may require further partition due to other setups.

Another approach is to increase L_1 (or $Y_{1,1,k}$ or both) enough to ensure that L_1 will not reach the most critical machine before the setup is finished.

All the methods above are not guaranteed to produce the minimal makespan. To do that we have to resort to the ILP model (see Appendix). The ILP model itself, however, can be approximated by an LP model, thus yielding a solution with polynomial complexity. If we use an LP model, we can obtain good integral solutions by the methods listed in Section 10.

10. Obtaining Locally Optimal Integral Solutions

To complete the model we still need to adapt the solution of the relaxed problem to the integrality constraints. We utilize two procedures: (i) **The Feasibility Procedure**, which starts with a given makespan and produces a feasible integral solution with that makespan--if one exists; and (ii) **The Optimizing Procedure**, which starts with the makespan of the relaxed problem and increases it sequentially until The Feasibility Procedure indicates success. The increments by which The Optimizing Procedure increases the makespan are designed to make sure that the minimal feasible makespan will never be exceeded. The complexity of the combined procedures is polynomial, and they are fast in practice.

We assume that the partition of the best relaxed solution is enforced. Our purpose in this section is to identify the *locally optimal integral solution*. That is, the optimal integral solution with the same operating procedure and number of lots and sub-lots. Since we do not temper with the partition, we can apply our procedures to each part separately. For convenience in presentation, we assume that the solution is unpartitioned.

Let MS^* denote the locally optimal integral solution (instead of the globally optimal integral solution as above). Then $MS^* - MS^r = h^* \geq 0$ is the difference in makespan due to the integrality constraints. Our objective is to find h^* and a feasible schedule which makes it possible to process the batch within $MS^r + h^*$. We start with the first operating procedure, and later solve for the second one.

10.1 Solving for the first operating procedure

To specify the solution we need to introduce The Feasibility and the Optimizing Procedures as applicable to the first operating procedure.

The Feasibility Procedure:

Input: Any non-negative value, h .

Output: Upon success, a feasible integral solution, with makespan $MS = MS^r + h$,
Otherwise, indication that $MS^* > MS^r + h$.

Iterations: For $i = 1$ to k , let

$$L_i = \min\{\text{INT}[(MS - T_n m + S_{i-1}(T_n - T_1) - \sum_{j=1, n-1} TT_j - SU_1) / \sum_{j=1, n-1} T_j], m - S_{i-1}\} \text{ (where } S_0 = 0\text{)};$$

$$S_i = S_{i-1} + L_i;$$

$$F_i = \text{FRACTION}[(MS - T_n m + S_{i-1}(T_n - T_1) - \sum_{j=1, n-1} TT_j - SU_1) / \sum_{j=1, n-1} T_j];$$

$$E_i = 1 - F_i \text{ (} E_i \text{ is required for The Optimizing Procedure)};$$

Success indication: $S_k = m$. ■

The procedure simply maximizes L_i subject to the constraint that the makespan should be MS. Theorem 1, i.e., specifying that M_1 and M_n should operate continuously, serves to maximize L_i here. Note that F_i and E_i are similar to f_i and \hat{e}_i in Theorem 3, but they are not identical.

The Optimizing Procedure:

1. Let $h_0 = 0, j = 1$;
2. iteratively, call The Feasibility Procedure with h_{j-1} ; upon success, set $h^* = h_{j-1}$ and STOP;
otherwise, set $h_j = h_{j-1} + \min_i \{E_i\} \sum_{j=1, n-1} T_j$; and start iteration $j+1$. ■

The basic idea behind The Optimizing Procedure is that if The Feasibility Procedure does not produce a feasible solution, there must exist some lot which should include at least one more item. Therefore, we look for the smallest addition to h which will cause one lot (at least) to increase by one item, and run The Feasibility Procedure again for the new h . We refer to the resulting values of h as **jump points**. A tip to the wise may be in order here: when programming The Optimizing Procedure add a small amount, say 10^{-6} , to h_j before calling The Feasibility Procedure. Otherwise, the jump point may be missed due to rounding errors.

The combined procedures' worst case complexity is polynomial. The proof is a direct extension of Theorem 5 in [17]. The procedures were programmed for the two machines case, and the numerical experience is that the optimum is usually achieved with less than $m/3$ iterations. The bound of Theorem 3 was usually at least twice that of the actual solution.

10.2 Solving for the second operating procedure

The complication here is that we have to adjust the sub-lots *and* the parent lots to be integral. A key observation we use to resolve this issue is that processing a parent lot under this scheme is an instance of processing a batch under MAXPARTIT. Therefore, if we know how many items are in a lot, we can apply the solution of 10.1 to each pair separately. This will provide the locally optimal integral sub-lot sizes.

We proceed to examine how many units can be included in L_i for any given makespan. Let Δt_i measure the time interval between starting L_i on M_1 and on M_n , then $x_i = \text{INT}(\Delta t_i / \sum_{k=1, n-1} Y_{i,1,k} T_k)$ is an upper bound on the number of items that L_i can comprise. The $Y_{i,1,k}$ values are obtained as per Section 7. When we run The Extended Feasibility Procedure (below), it is possible to compute Δt_i for each lot. Hence, we can simply try to fit x_i units in L_i , and if necessary decrease it to $x_i - 1$, $x_i - 2$, and

so on. The largest feasible value is the one we specify. Finally, let $MS_i^*(x)$ be the minimal makespan for a batch of x units under **MAXPARTIT** with the same number of lots as the number of sub-lots in parent lot i between the same machines, and we are ready to state the procedure.

The Extended Feasibility Procedure:

Input: Any non-negative value, h .

Output: Upon success, a feasible integral solution, with makespan $MS = MS^r + h$,
Otherwise, indication that $MS^* > MS^r + h$.

Iterations: For $i = 1$ to k , let

$$\Delta t_i = MS - T_n m + S_{i-1}(T_n - T_1) - \sum_{j=1, n-1} TT_j - SU_1;$$

$$L_i = \max\{\text{INT}(\Delta t_i / \sum_{k=1, n-1} Y_{i,1,k} T_k), m - S_{i-1}\};$$

REPEAT: if $MS_i^*(L_i) > \Delta t_i + L_i T_n$ then $L_i = L_i - 1$;

UNTIL: $MS_i^*(L_i) \leq \Delta t_i + L_i T_n$;

$$S_i = S_{i-1} + L_i \text{ (where } S_0 = 0\text{);}$$

$$E_i = MS_i^*(L_i + 1) - \Delta t_i - L_i T_n \text{ (} E_i \text{ is required for The Optimizing Procedure);}$$

Success indication: $S_k = m$. ■

The Extended Optimizing Procedure is almost identical to the preceding version. The only difference is that when updating h we use $h_j = h_{j-1} + \min\{E_i\}$, i.e., we do not multiply $\min\{E_i\}$ by $\sum_{j=1, n-1} T_j$.

Though slightly more complicated than the algorithm for the first operating procedure, the algorithm here is still polynomial and tractable.

In fact, our method is actually too good in a sense, because in practice we'll need some time buffers (see [14, 17]), to accommodate fluctuations in the processing rate etc. These buffers will probably be large relative to the accuracy of the procedure. Therefore, it makes sense to let the increment in h be "too large."

11. Modifications

So far we assumed that the cost of each transfer is a constant, regardless of the quantity transferred. We did not treat the issue of work-in-process explicitly. That is, we tacitly assumed there is enough storage space near each machine for any lot size. We did not consider resources that require the same time to process any lot size--such as ovens. Finally, we assumed that the product is processed on a single linear

sequence of machines. To adapt the model for implementation we may have to deal with some or all of these issues. In this section we outline how to do this.

Suppose the real cost of a transfer from M_i is of the form $a_i + b_i L$, where a_i and b_i are positive constants and L is the lot size. Then summing for all lots, our cost is $a_i k + b_i m$. $b_i m$ is fixed for the batch, and a_i takes the place of our transferring cost, C_i . Therefore, all we have to do is reduce the budget by $m \sum b_i$, and our model still applies. Usually we can approximate the real transferring costs by such a function, so our model is not restrictive here.

Next, let us discuss the WIP. There are two issues involved: (i) the money invested in this inventory, and (ii) congestion in the plant. The money invested in WIP is only an issue if the raw materials and purchased components for some of the items of the batch can be acquired during the processing. If this is the case, we should consider using smaller batches. If we insist on large batches, however, we can accommodate a restriction on the rate of investment in WIP by specifying a dummy machine, M_0 , in front of M_1 . A transfer from M_0 will cost C_0 , reflecting the fixed transaction cost of ordering/receiving the materials plus the cost of releasing them to production. If the speed assigned to M_0 is not less than that of the slowest machine, the makespan can still be minimized as before. *This may require a larger budget, however.*

It is possible to use the model to choose the best speed for M_0 so the total cost of WIP and transferring will be minimized for any feasible makespan. Next, if we have the value of a savings of a time unit in the makespan, we can minimize the total WIP, transferring, and makespan cost. This will require using the model as input for a search procedure that will search for the optimal T_0 . Note that if T_0 is set equal to $T_p = \text{Max}\{T_i\}$, then at least between M_0 and M_p the model will call for equal parent lots. The sub-lots are still likely to vary, however.

If we can sell the first items of the batch before finishing the processing, then it makes sense to allow M_n to work intermittently too. We can do this by a symmetric dummy machine, M_{n+1} . Again we can optimize T_{n+1} similarly to the case of T_0 .

If the WIP is a problem due to congestion, the formal problem becomes tough mathematically. We can handle it in practice by dividing the batch and the budget to (roughly) equal sub-batches, and solving for each sub-batch by our model. Since the sub-batches are equal, then for every number of sub-batches we have to solve the model once, and check the solution for congestion. As a rule, we should divide the batch to the largest possible sub-batches which do not cause congestion.

Next we discuss special resources such as ovens, which take the same time to process a lot regardless of its size. A convenient way to deal with these is to model them as transfers, rather than as

machines. This raises a sub-problem: make sure that the resource will be available for the next lot in time (i.e., the lots must not be too small). The sub-problem can be solved by modifying a model presented in [17, Section 5], where the number of vehicles is limited. Our special resource acts as such a vehicle, connecting the two adjacent machines. [17]'s model can also serve if there are j such resources in parallel, or if the resource has a limited capacity.

When assembling a batch of products, the assembly operation may be fed by more than one line of machines. The problem is to coordinate these lines to feed the assembly on time. To solve this problem we can model the assembly as the last machine (M_n) for all the lines feeding it. This creates an opportunity to optimize the transportation budget allocation to the sequences. If the objective function is to minimize the project makespan, this can be done by The Greedy Heuristic. At each stage, only transfers which decrease the makespan of the *longest* sequence are considered.

12. Conclusion

We developed fast solution techniques for the single job-several machines transfer lot sizing problem. The techniques can be implemented in new or existing MRP packages. They are easy to program, and do not require support by additional mathematical programming modules. We showed that by allowing the lot sizes to vary, the number of necessary transfers tends to be $O(\log m)$, and that by allowing intermediary machines to idle intermittently the makespan can be decreased considerably. We conclude the paper with a partial list of open research questions (see [17] for other open questions).

- Determine the complexity of the problem; i.e., find a solution in P or prove NP-completeness.
- Develop more heuristics, as long as no efficient polynomial solution is found. For instance, in the second operating procedure we may allow intermittent idling within each parent lot, or allow **sub-parent lots**. That is, *use the first or the second operating procedure within the second operating procedure*.
- Generalize the problem for several jobs in a flow shop environment (see [1] and [17] for preliminary results).
- Investigate the implications for the Job Shop Scheduling Problem (not necessarily a flow shop); combined heuristics.
- Relax the assumption that the production rates are deterministic and known exactly. This issue includes the problem of obtaining the best estimators for the true rates of production to minimize the expected makespan. (See [17] for some basic sensitivity analysis results which can be extended to the present model.)

- Consider the case where several operations are required on the same machine, calling for intermediary setups.
- Introduce multidimensional budget constraints, e.g., manpower and equipment.

REFERENCES:

- [1] Baker, Kenneth R., Lot Streaming to Reduce Cycle Time in a Flow Shop, Working Paper #203, The Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH 03755, June 1987.
- [2] Baker, Kenneth R. and David F. Pyke, Algorithms for the Lot Streaming Problem, Working Paper #233, The Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH 03755, August 1988.
- [3] Bellman, R., A. O. Esogbue and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, 1982.
- [4] Chase, Richard B. and Nicholas J. Aquilano, *Production and Operations Management*, 5th Edition, Richard D. Irwin, Inc., Homewood, Illinois, 1989.
- [5] Coffman, E. G., Jr. (Ed.) *Computer and Job-Shop Scheduling Theory*, Coauthored by J. L. Bruno, E. G. Coffman, Jr., R. L. Graham, W. H. Kohler, R. Sethi, K. Steiglitz and J. D. Ullman, John Wiley, 1976.
- [6] Dobson, Gregory, Uday S. Karmarkar and Jeffrey L. Rummel, Batching to Minimize Flow Times on One Machine, *Management Science*, 33, #6, 1987, pp. 784-799.
- [7] Goldratt, Eliyahu and Robert E. Fox, *The Race*, North River Press, Box 241, Croton-on-Hudson, 1986.
- [8] Graves, Stephen C. and Michael M. Kostreva, Overlapping Operations in Material Requirements Planning, *Journal of Operations Management*, 6, #3, 1986, pp. 283-294.

- [9] Hall, Robert W., *Driving the Productivity Machine: Production Planning and Control in Japan*, American Production and Inventory Control Society, 1981.
- [10] Jacobs, F. Robert, OPT Uncovered: Many Production Planning and Scheduling Concepts Can be Applied With or Without the Software, *Industrial Engineering*, 16, #10, 1984.
- [11] Lee, Sang M., Lawrence J. Moore and Bernard W. Taylor, *Management Science*, 2nd Edition, Wm. C. Brown Publishers, Dubuque, Iowa, 1985., pp. 330-334.
- [12] McLeavey, Dennis W. and Seetharama L. Narasimhan, *Production Planning and Inventory Control*, Allyn and Bacon, Inc., 1985.
- [13] Orlicky, Joseph, *Material Requirements Planning*, McGraw-Hill, New York, 1975.
- [14] Ronen, Boaz, "Optimal Time Buffers in Synchronized Manufacturing Environments," Working Paper, New York University, 1987.
- [15] Ronen, Boaz and Martin K. Starr, "Synchronized Manufacturing as in OPT: From Practice to Theory," *Computers and Industrial Engineering* (to appear).
- [16] Shingo, Shigeo, *A Revolution in Manufacturing: The SMED System*, Productivity Press, Stamford, Connecticut, 1985.
- [17] Trietsch, Dan, Optimal Transfer Lots for Batch Manufacturing On Several Machines, Working Paper, February and November 1987, revised July 1989.
- [18] Truscott, William G., "Scheduling Production Activities in Multi-Stage Manufacturing Systems," *International Journal of Production Research*, 23, #2, 1985, pp. 315-328.
- [19] Truscott, William G., "Production Scheduling with Capacity Constrained Transportation Activities," *Journal of Operations Management*, 6, #3, 1986, pp. 333-348.
- [20] Vollmann, Thomas E., "OPT as an Enhancement to MRP II," *Production and Inventory Management*, 2nd Quarter, 1986, pp. 38-46.

APPENDIX:

This appendix lists the formulation of the problem as an ILP model, and supplies proofs and details omitted in the main body of the paper.

ILP Formulation of Problem (P): Let $t_{i,j}$ be the time item j ($j = 1, 2, \dots, m$) is transferred to M_{i+1} ($i = 1, 2, \dots, n-1$), and $t_{n,j}$ is the time item j finishes processing on M_n . Let $y_{i,j} = 1$ if $t_{i,j}$ coincides with finishing the processing of item j on M_i , and $y_{i,j} = 0$ if item j is held until at least one additional item is processed. This leads to the following ILP formulation:

$$\min t_{n,m}$$

s.t.

$$t_{i,j} \geq jT_i + SU_i ; \text{ for all } i, j \quad (A1)$$

$$t_{i,j} \geq t_{i-1,j-k} + (k+1)T_i + TT_{i-1} ; i=2,3,\dots,n, \text{ for all } j, k = 0, 1, \dots, j-1 \quad (A2)$$

$$t_{i,j} \geq t_{i,j+1} - mT_i y_{i,j} ; \text{ for all } i, j=1,2,\dots,m-1 \quad (A3)$$

$$\sum_{i=1,n-1} C_i \sum_{j=1,m} y_{i,j} \leq B \quad (A4)$$

$$y_{i,j} \in \{0, 1\}.$$

We have about $nm^3/3$ constraints (predominantly (A2)'s), and most of them will be lax. (A1) takes care of the setups. With $k = 0$, (A2) ensures that all items will be transferred from M_{i-1} prior to being processed by M_i ; and with $k \geq 1$, it ensures that item j will not be processed before items $j-1, j-2$, and so on. (A3) is lax if $y_{i,j} = 1$, i.e., if a transfer follows the processing of item j on M_i immediately; if $y_{i,j} = 0$, (A3) implies $t_{i,j} \geq t_{i,j+1}$, and due to the target function this will be satisfied as an equality. Finally, (A4) is our budget constraint. Note that the number of constraints involved is polynomial, but rather large, so the ILP approach may not be attractive in practice.

For a flow shop environment, it is straightforward to generalize this ILP model for several consecutive jobs.

Theorem 2: *If the feasibility conditions are satisfied, $[(5)] \geq 1$ AND $[(5)]Q^{k-1} \geq 1$, then $L_1^* = [(5)]$; else, if the feasibility conditions are satisfied, then The Adjustment Procedure yields an optimal solution.*

Proof: We first show that (3) and (5) yield the best super-relaxed solution. If M_n can start upon receipt of L_1 , the makespan is $L_1(T_1 + T_2 + \dots + T_{n-1}) + mT_n$; i.e., the time required for the first lot to reach M_n plus the time required by M_n to finish the batch. By symmetry, if M_n is ready in time for the last lot, the makespan is $L_k(T_2 + T_3 + \dots + T_n) + mT_1$. Clearly (3) and (5) ensure $L_1^*(T_1 + T_2 + \dots + T_{n-1}) + mT_n = L_k^*(T_2 + T_3 + \dots + T_n) + mT_1$. We proceed to prove by

induction that any other lots are not optimal:

(a) Let $k = 2$. If $L_1 > L_1^*$, then the makespan is at least $L_1(T_1 + T_2 + \dots + T_{n-1}) + mT_n > L_1^*(T_1 + T_2 + \dots + T_{n-1}) + mT_n$. Alternately, if $L_1 < L_1^*$, then $L_2 > L_2^*$, and the makespan is at least $L_2(T_2 + T_3 + \dots + T_n) + mT_1 > L_2^*(T_2 + T_3 + \dots + T_n) + mT_1$.

(b) Let $k \geq 3$, and the induction assumption is that (3) and (5) are optimal for $k-1$ lots. For $k = 3$, we proved the induction assumption in (a). If $L_1 > L_1^*$, then the proof in (a) holds here too, and the makespan will be larger than for L_1^* . If $L_1 < L_1^*$, then by the optimality criterion of Bellman, the best we can do downstream is solve the $k-1$ case for the remaining $m-L_1$ items. This implies $L_j = L_j^*(m-L_1)/(m-L_1^*) > L_j^*$; for all $j = 2, 3, \dots, k$, i.e., $L_k > L_k^*$, and the proof in (a) holds again. (Note that in the case for which we applied the optimality criterion of Bellman, M_2 and the downstream machines will have to idle between L_1 and L_2 .)

It remains to show that The Adjustment Procedure, which we use if $[(5)] < 1$ OR $[(5)]Q^{k-1} < 1$ (i.e., the super-relaxed solution is not also the regular relaxed solution), preserves the optimality of the solution. We concentrate on Part (a) of the procedure, Part (b) being symmetric. M_n cannot start before time $\sum_{j=1, n-1} T_j$, so by feeding it at this time *and* making sure it can operate continuously until it finishes the batch, the makespan will be minimized. Except for Step 4, this is the case here. As for Step 4, it may force some of the last items to be transferred from M_1 toward M_n one-by-one, *as soon as they are finished*. In addition, if Step 4 reduces L_{j-1} to increase L_j , then L_{j-1} will be released for transfer sooner than scheduled before. All this cannot cause any delay relative to any feasible solution, so it preserves the optimality. Finally, the stopping criterion that is incorporated in Step 4 is valid because the original lots are non-increasing. ■■■

Proofs Regarding Heuristics 1 and 2:

We now prove that Heuristic 1 yields a feasible solution that increases the relaxed makespan by $\text{Max}\{e_i\}\sum_{j=1, n-1} T_j$. First note that $S_k = m$, so it does not require rounding. Next, if some of the last lots contain one unit each, as a result of Step 4 in The Adjustment Procedure, then the corresponding S_i values are integers as well, and do not require rounding either. Therefore, rounding any S_i value up cannot cause any subsequent lot to be less than 1. Hence, the lot sizes are feasible. Now, if we start processing under this new scheme, M_n will have to wait $e_i\sum_{j=1, n-1} T_j$ for L_1 (relative to the relaxed solution), or $\text{Max}\{e_i\}\sum_{j=1, n-1} T_j$ at most. This completes the proof.

The proof that Heuristic 2 yields a feasible solution that increases the relaxed makespan by $\text{Max}\{f_i\}_{\Sigma_{j=2,n} T_j}$ is by symmetry: rounding up for the original problem truncates the symmetric problem; $\Sigma_{j=2,n} T_j$ assumes the role of $\Sigma_{j=1,n-1} T_j$; and f_i replaces e_i . ■■■

Theorem 4: For any partition $\{p(0)=1, p(1), p(2), \dots, p(r), p(r+1)=n\}$ (feasible or not), $\text{MMR}_{1,n} \geq \Sigma_{i=0,r} \text{MMR}_{p(i),p(i+1)}$.

Proof: Substituting from (1) and (7), we need to show that

$$\Sigma_{i=1,n} T_i - \text{Max}_{i=1,n} \{T_i\} \geq \Sigma_{s=0,r} \Sigma_{i=p(s),p(s+1)} T_i - \Sigma_{s=0,r} \text{Max}_{i=p(s),p(s+1)} \{T_i\}. \quad (\text{A5})$$

There exists an index w ($0 \leq w \leq r$) such that $\text{Max}_{i=1,n} \{T_i\} = \text{Max}_{i=p(w),p(w+1)} \{T_i\}$. Therefore the right hand side of (A5) can be written as

$$\begin{aligned} & \Sigma_{i=1,n} T_i + \Sigma_{s=1,r} T_{p(s)} - \Sigma_{s=0,r} \text{Max}_{i=p(s),p(s+1)} \{T_i\} = \\ & \Sigma_{i=1,n} T_i + \Sigma_{s=1,r} T_{p(s)} - \Sigma_{s=0,w-1} \text{Max}_{i=p(s),p(s+1)} \{T_i\} - \Sigma_{s=w+1,r} \text{Max}_{i=p(s),p(s+1)} \{T_i\} - \text{Max}_{i=1,n} \{T_i\}. \end{aligned}$$

Subtract $\Sigma_{i=1,n} T_i - \text{Max}_{i=1,n} \{T_i\}$ from both sides, and it remains to show

$$\Sigma_{s=1,w-1} T_{p(s+1)} - \Sigma_{s=0,w-1} \text{Max}_{i=p(s),p(s+1)} \{T_i\} + \Sigma_{s=w+1,r} T_{p(s)} - \Sigma_{s=w+1,r} \text{Max}_{i=p(s),p(s+1)} \{T_i\} \leq 0.$$

But for any $1 \leq s \leq w-1$, $T_{p(s+1)} - \text{Max}_{i=p(s),p(s+1)} \{T_i\} \leq 0$, and similarly for any $w+1 \leq s \leq r$,

$$T_{p(s)} - \text{Max}_{i=p(s),p(s+1)} \{T_i\} \leq 0, \text{ and the theorem follows. } \blacksquare$$

Lemma 1: Let (i, j) be any pair such that $i < p < j$, and such that $T_p > \text{Max}\{T_i, T_j\}$, then (i, j) is an infeasible pair.

Proof: Define $\text{SUM}_1 = \Sigma_{s=i+1,p-1} T_s$ and $\text{SUM}_2 = \Sigma_{s=p,j-1} T_s$, then

$Q_{i,p} = (\text{SUM}_1 + T_p) / (\text{SUM}_1 + T_i)$ and $Q_{i,j} = (\text{SUM}_1 + \text{SUM}_2 + T_j) / (\text{SUM}_1 + \text{SUM}_2 + T_i)$. We proceed to check if (i, j) is a feasible pair, and clearly if so then we must have $Q_{i,j} \geq Q_{i,p}$. After some algebra we get $Q_{i,j} \geq Q_{i,p} \iff T_i(\text{SUM}_1 + \text{SUM}_2 + T_j) \geq T_p(\text{SUM}_1 + \text{SUM}_2 + T_i)$, but $T_i < T_p \iff T_i(\text{SUM}_1 + \text{SUM}_2) < T_p(\text{SUM}_1 + \text{SUM}_2)$, and $T_j < T_p \iff T_i T_j < T_p T_i$. Hence $Q_{i,p} > Q_{i,j}$, and (i, j) is infeasible. ■■■

Theorem 5: Let $\text{MINPARTIT} = \{p(0)=1, p(1), p(2), \dots, p(r), p(r+1)=n\}$, then

$$\text{MMR}_{1,n} = \Sigma_{i=0,r} \text{MMR}_{p(i),p(i+1)}.$$

Proof: By observing the proof of Theorem 4, $\text{MMR}_{1,n} = \Sigma_{s=0,r} \text{MMR}_{p(s),p(s+1)}$ if and only if

$$T_{p(i+1)} = \text{Max}_{s=p(i),p(i+1)} \{T_s\}; \text{ for all } i = 0, 1, \dots, w-1, \text{ AND } T_{p(i)} = \text{Max}_{s=p(i),p(i+1)} \{T_s\};$$

for all $i = w+1, w+2, \dots, r$, where $p(w)$ is the index of the slowest machine. Now, if the only partition is at the slowest machine, the theorem is satisfied trivially. By Lemma 1 $M_{p(w)}$ is part of the partition, so any other machine in the partition must either precede it or follow it. We concentrate on the former,

and look at $M_{p(i)}$ for some $0 \leq i < w$. We have to show $T_{p(i+1)} \geq T_{p(i)}$, and then by Lemma 1 we'll have shown that $T_{p(i+1)} = \text{Max}_{s=p(i), p(i+1)} \{T_s\}$ as required. If $i+1 = w$, then this is clearly true, so we assume $i < w-1$. But by construction of the minimal partition we have $Q_{p(i), p(i+1)} > Q_{p(i), p(w)} \geq 1$ (since $T_{p(w)} \geq T_{p(i)}$) $\implies T_{p(i+1)} > T_{p(i)}$.

This completes the proof for $1 \leq i < w$. As for $w < i \leq r+1$, this side follows by symmetry. ■■■

In order to prove Theorem 6, we need two additional lemmas, not listed in the main body of the paper.

Lemma A1: Let $1 \leq p(s-1) < p(s) < j \leq n$, where $(p(s-1), p(s))$ is a pair in MINPARTIT, then $Q_{p(s), j} < Q_{p(s-1), p(s)}$.

Proof: Let $\text{SUM}_1 = \sum_{i=p(s-1)+1, p(s)-1} T_i$, and $\text{SUM}_2 = \sum_{i=p(s)+1, j-1} T_i$, then the lemma states:

$$(\text{SUM}_2 + T_j) / (\text{SUM}_2 + T_{p(s)}) < (\text{SUM}_1 + T_{p(s)}) / (\text{SUM}_1 + T_{p(s-1)}).$$

We proceed by negation, i.e., assume

$$\begin{aligned} (\text{SUM}_2 + T_j) / (\text{SUM}_2 + T_{p(s)}) &\geq (\text{SUM}_1 + T_{p(s)}) / (\text{SUM}_1 + T_{p(s-1)}) \implies \\ \text{SUM}_1 T_j + \text{SUM}_2 T_{p(s-1)} + T_j T_{p(s-1)} &\geq \text{SUM}_1 T_{p(s)} + \text{SUM}_2 T_{p(s)} + (T_{p(s)})^2. \end{aligned} \quad (\text{A6})$$

Now look at $Q_{p(s-1), j} - Q_{p(s-1), p(s)}$. By the definition of MINPARTIT, it must be strictly negative. That is

$$\begin{aligned} (\text{SUM}_1 + T_{p(s)} + \text{SUM}_2 + T_j) / (\text{SUM}_1 + T_{p(s)} + \text{SUM}_2 + T_{p(s-1)}) - \\ (\text{SUM}_1 + T_{p(s)}) / (\text{SUM}_1 + T_{p(s-1)}) < 0 \implies \\ (\text{SUM}_1 + T_{p(s)} + \text{SUM}_2 + T_j)(\text{SUM}_1 + T_{p(s-1)}) - \\ (\text{SUM}_1 + T_{p(s)} + \text{SUM}_2 + T_{p(s-1)})(\text{SUM}_1 + T_{p(s)}) < 0. \end{aligned}$$

By simple algebra, this reduces to

$$\text{SUM}_1 T_j + \text{SUM}_2 T_{p(s-1)} + T_j T_{p(s-1)} < \text{SUM}_1 T_{p(s)} + \text{SUM}_2 T_{p(s)} + (T_{p(s)})^2,$$

thus directly contradicting (A6). ■■■

Corollary A1: Under MINPARTIT, the series $\{Q_{p(i), p(i+1)}\}_{i=0, r}$ is monotone decreasing. ■■■

Lemma A2: Under the conditions of Lemma A1, let $p(s-1) < i < p(s)$, then $Q_{i, p(s)} > Q_{p(s), j}$.

Proof: By Lemma A1, $Q_{p(s), j} < Q_{p(s-1), p(s)}$, so it suffices to show

$$Q_{i, p(s)} \geq Q_{p(s-1), p(s)}. \quad (\text{A7})$$

By the definition of MINPARTIT,

$$Q_{p(s-1),i} \leq Q_{p(s-1),p(s)}. \quad (A8)$$

Now look at the symmetric problem, and the symmetric MINPARTIT is the original MINPARTIT, though listed in reversed order. This is true because feasibility of pairs is unaltered under symmetry, and MINPARTIT is essentially the set of the largest possible feasible pairs. In the symmetric problem, $Q_{p(s-1),p(s)}$ is replaced by $1/Q_{p(s-1),p(s)}$, which applies to the pair $(p(s), p(s-1))$. Therefore, by symmetry to (A8) we obtain $1/Q_{i,p(s)} \leq 1/Q_{p(s-1),p(s)}$, which leads directly to (A7). ■■■

Theorem 6: *Let PARTIT be any feasible partition which is not identical to MINPARTIT then MINPARTIT must be properly contained in PARTIT.*

Proof: Let $\text{MINPARTIT} = \{p(s)\}_{s=0,r+1}$, and we use simple indices such as i, j for machines in PARTIT which are not in MINPARTIT. First, let us show (by negation) that PARTIT cannot be properly contained in MINPARTIT. Suppose $1 \leq s \leq r$ is the smallest index such that $p(s)$ belongs to MINPARTIT but not to PARTIT, and let $p(t)$ such that $s < t \leq r+1$, be the smallest index of a machine which belongs to MINPARTIT and to PARTIT (recall $p(r+1)=n$, and M_n is included in all partitions, therefore, such a t must exist). By construction of MINPARTIT, $Q_{p(s-1),p(t)} < Q_{p(s-1),p(s)}$, and hence $(p(s-1), p(t))$ is not a feasible pair. This contradicts the assumption that PARTIT is feasible and contained in MINPARTIT.

Assume then that PARTIT includes at least one machine which does not belong to MINPARTIT. Pick the machine with the smallest index, say i , which belongs to PARTIT but not to MINPARTIT, and clearly $i > 1$ (since $M_1 = M_{p(0)}$ belongs to MINPARTIT); therefore there exists an index s such that $1 < s \leq r+1$, $p(s-1) < i$, and $p(s) > i$. Let $p(k)$ be the index of the machine paired with i from below, i.e., the pair $(p(k), i)$ is in PARTIT.

We now show that $p(s-1)$ is $p(k)$. By construction of MINPARTIT, if $p(k+1) \leq p(s-1)$ then $Q_{p(k),i} < Q_{p(k),p(k+1)}$ and thus $(p(k), i)$ is an infeasible pair, contradicting the feasibility of PARTIT. Hence $p(s-1)$ must be in PARTIT. By symmetry, it is clear that if j is the last machine in PARTIT - MINPARTIT, then all the machines with a larger index in MINPARTIT must also be in PARTIT. It remains to show that no intermediate machines in MINPARTIT are strictly within feasible pairs of machines in PARTIT (where each pair may include up to one machine which is also in MINPARTIT; we dealt with the case where both of them are in MINPARTIT above by showing that PARTIT cannot be properly contained in MINPARTIT). Now, if any other index, say i' , exists in PARTIT such that $i < i' < p(s)$, take the largest such i' , and rename it as i . Hence, i is now the largest index of a machine in PARTIT - MINPARTIT such that $p(s-1) < i < p(s)$. Also, we know that $p(s-1)$ is in PARTIT. We proceed to prove that $p(s)$ must be in PARTIT as well. To that end it suffices to show that $Q_{i,p(s)} = \text{Max}_{j>p(s)}\{Q_{i,j}\}$ (i.e., no feasible pair

exists with $p(s)$ strictly within it). This result is assured by Lemma A2. Now, look for the next machine in **PARTIT - MINPARTIT** which can again be called j . We just proved that the machine in **MINPARTIT** nearest to i from above must be in **PARTIT**. By symmetry, the nearest machine in **MINPARTIT** to j from below must be in **PARTIT** also, and it follows that any machines in **MINPARTIT** between those two are also in. Now, rename j as i and repeat the whole procedure until no machines are found in **PARTIT - MINPARTIT**. ■■■

Lemma 2: Let (i, j) be a pair such that $T_k \leq \min\{T_i, T_j\}$; for all $i < k < j$, then (i, j) is a feasible pair.

Proof: Let $SUM = \sum_{s=i+1, k-1} T_s$ and assume $Q_{i,j} \geq 1$, then

$Q_{i,k} = (SUM + T_k)/(SUM + T_i) \leq 1$ (since $T_k \leq T_i$). $\implies Q_{i,k} \leq Q_{i,j}$; for all k , and the lemma is satisfied. If $Q_{i,j} < 1$, the proof is by symmetry. ■■■

Theorem 7: The series $\{MR_{i,j}(k)\}_{k=2,3,\dots,m}$ is monotone decreasing.

Proof: For convenience, we use Q for $Q_{i,j}$ where there is no risk of confusion. $MR_{i,j}(k)$ is simply the difference between L_1 for $k-1$ and for k transfers multiplied by $\sum_{s=i,j-1} T_s$. Assuming $K_{i,j} > k \geq 2$ we obtain:

$$\begin{aligned} MR_{i,j}(k)/\sum_{s=i,j-1} T_s &= L_1 |k-1 \text{ transfers} - L_1 |k \text{ transfers} = \\ &= m/(1+Q+Q^2+\dots+Q^{k-2}) - m/(1+Q+Q^2+\dots+Q^{k-1}) = \\ &= mQ^{k-1}/[(1+Q+Q^2+\dots+Q^{k-2})(1+Q+Q^2+\dots+Q^{k-1})]. \end{aligned} \quad (A9)$$

$MR_{i,j}(K_{i,j})$ is bounded from above by the value indicated for it by (A9), so it is enough to show that (A9) leads to a monotone decreasing series. Assume now that $Q \leq 1$. Under this assumption the numerator in (A9) is monotone non-increasing and the denominator is monotone increasing with k . Hence, $MR_{i,j}(k) < MR_{i,j}(k-1)$ as required. The proof for $Q > 1$ follows by symmetry, since in the symmetric problem $Q < 1$, but the series $\{MR_{i,j}(k)\}$ is identical. ■■■

Note that (8) can be developed directly from (A9) by using the geometric sum formula where applicable.

Theorem 8: $MR_{i,j}(2) > MMR_{i,j}/2$.

Proof: Assume $T_i \geq T_j$, $\implies Q = Q_{i,j} = \sum_{k=i+1,j} T_k / \sum_{k=i,j-1} T_k \leq 1$ and $MMR_{i,j} = (m-1)\sum_{k=i+1,j} T_k$.

By (8), $MR_{i,j}(2) = m(1 - 1/(1+Q))\sum_{k=i,j-1} T_k$. Since $Q \leq 1$,

$$2MR_{i,j}(2) \geq m\sum_{k=i,j-1} T_k \geq m\sum_{k=i+1,j} T_k > (m-1)\sum_{k=i+1,j} T_k = MMR_{i,j}.$$

By symmetry, the same result holds if $T_i < T_j$. ■■■

Partitioning the Problem by Dynamic Programming

Let:

- $FB = B - \sum C_i$, be the total free budget after accounting for the essential transfers (i.e., one transfer for each pair $(i, i+1)$).
- F = the free budget remaining for allocation at any stage; e.g., at the first stage, $F = FB$.
- $TG_i(F, j, k)$ = the total makespan reduction from M_i and downstream if we indicate k transfer lots from M_i to M_j , and use the rest of the free budget from M_j and downstream *optimally*. (In this definition and the following ones we assume (i, j) is a feasible pair.)
- $TG_j^*(F)$ = the total makespan reduction possible from M_j and downstream if we have a free budget of F at M_j .
- $TR_{i,j}(k)$ = the total makespan reduction accumulated for pair (i, j) using k transfers between i and j , then,
 $TR_{i,j}(1) = 0$ (the first transfer is essential),
 $TR_{i,j}(k) = TR_{i,j}(k-1) + MR_{i,j}(k)$; $2 \leq k < K_{i,j}$,
 $TR_{i,j}(K_{i,j}) = MMR_{i,j}$.
- $LABEL1_i(F)$ = the number of transfers required from M_i when the free budget remaining there is F , to achieve the optimal makespan reduction indicated by $TG_i^*(F)$.
- $LABEL2_i(F)$ = the index of the machine paired directly to M_i when we have a free budget of F there, to realize the optimal makespan reduction indicated by $TG_i^*(F)$ from M_i and downstream.

We are now ready to state our recursion formulae. First we have

$$TG_{n-1}^*(F) = TR_{n-1,n}(\text{Min}\{\text{INT}(F/C_{n-1}), K_{n-1,n}\}). \quad (A10)$$

Next, for stage i , assume we have $TG_j^*(F)$ from the former recursions for all $j > i$ (for $i = n-2$ we have (A10)), then for any j such that (i, j) is a feasible pair we have

$$\begin{aligned} TG_i(F, j, 1) &= TG_j^*(F) ; \text{ for all } 0 \leq F \leq FB, \\ TG_i(F, j, k) &= TR_{i,j}(k) + TG_j^*(F - (k-1)\sum_{s=i,j-1} C_s) ; \text{ any } F, 2 \leq k \leq K_{i,j}, \\ TG_i^*(F) &= \text{Max}_{j,k} \{TG_i(F, j, k)\}. \end{aligned}$$

And thus we can compute all the $TG_i^*(F)$ values, including $TG_1^*(FB)$. Finally, to trace the optimal allocation from M_i and downstream, we compute the $LABEL1_i(F)$ and $LABEL2_i(F)$ values during the recursion by

$$(LABEL1_i(F), LABEL2_i(F)) = \text{args}\{TG_i(F, LABEL1_i(F), LABEL2_i(F)) = TG_i^*(F)\}.$$

The Complexity of the DP Partitioning Algorithm

Assume we use tables where the row k corresponds to utilizing k transfers in the next stage, and each column corresponds to a free budget value. Therefore each table has $O(F)O(\text{max \# of transfers})$ entries. There are at most $n(n-1)/2$ feasible pairs (at least $n-1$, but this is not important for the worst case analysis), which can be identified in $O(n^3)$. Thus we have to build $O(n^2)$ tables, each based on up to $n-1$ possible routes to feasible downstream pair-mate machines. This leads us to $O(n^3)O(F)O(\text{max \# of transfers})$. Generally $O(F)$ depends on the budget, but cannot exceed $O(m)$. As for the max # of transfers, if $Q_{i,j}$ is likely to be 1, we have to consider up to m transfers, leading to $O(n^3m^2)$. If $Q_{i,j}$ is not likely to be 1 more than a bounded number of times which is not dependent on m , than by (11) and (12) we know that $K_{i,j}$ is $O(\log m)$, leading to $O(n^3m \log m)$.

Note that by using Theorem 6, we can save a lot of effort when looking for all possible feasible pairs that include M_i . The theorem allows us to confine such searches to the subset to which i belongs in MINPARTIT. Furthermore, we can partition the problem to the parts implied by MINPARTIT, and use a similar master program to assign the budget to the parts. The advantage is realized if FB is larger than the budget which can be utilized in some single parts. For instance, if $M_{p(s)}$ and $M_{p(s+1)}$ belong to MINPARTIT, the optimal solution cannot specify spending more in this part than the cost of $K_{p(s),p(s+1)}$ compound transfers, which may be significantly less than FB . Assigning the budget to the parts can be solved as a simple instance of the dynamic programming knapsack model (e.g., see [4]). In fact our algorithm above is a direct extension of this classic model.

Calculating L_i and $Y_{i,j,k}$ for the second operating procedure

We now discuss in more detail how to calculate L_i and $Y_{i,j,k}$. We follow the schematic outline of Section 7 step by step. We repeat the outline below, for convenience.

- (1) At stage i , given L_i^t and $Y_{p,j,k}$ for $p < i$ and all j, k , calculate $Y_{i,j,k}$ for all j, k ; if $i = K$, go to (3); else, set $i = i + 1$ and go to (2);
- (2) Given L_p^t and $Y_{p,j,k}$ for $p < i$ and all j, k , calculate L_i^t and return to (1)
- (3) For $i = 1$ to K let $L_i = mL_i^t / \Sigma L_p^t$

Step 1: By (13) and (14) we know exactly when L_i^t starts on M_1 ($ST_{i,1}$), and when its first sub-lot is due at M_n ($ST_{i,n}$). Then the problem of finding the values for $Y_{i,j,k}$ can be solved by applying the super-relaxed solution of the two machines model recursively. That is, if parent lot i is sub-divided to k sub-lots when processed by M_j , we use (5) and (3) with $Q = T_{j+1}/T_j$ and $m = 1$ (since the Y values sum to 1, rather than to m). This policy, if feasible, is optimal when the number of sub-lots for each machine is given. The optimality follows directly from the optimality of the two machines model solution. Any other choice of the $Y_{i,j,k}$ values will lead to unnecessary delays.

Step 2: Observing the solution for the Y values as discussed in the preceding paragraph, we note that they are invariant with L_i^t . In contrast, the time elapsed between $ST_{i,1}$ and the instant the first sub-lot of L_i can reach M_n is a function of L_i , namely $L_i \Sigma_{k=1,n-1} (Y_{i,1,k} T_k)$. Since we have $ST_{i,n}$, we know the time allotted for this purpose, which is $[(14)] - [(13)]$. On the one hand, if the value we get by the tentative L_i is less than the allotted time, then we could have processed more items for the same transfer costs. Hence, L_i should be larger. On the other hand, by Theorem 1 we know that M_n does not have to wait. Hence L_i must be exactly large enough to use up all the time allotted to it by $[(14)] - [(13)]$. That is

$$L_i = (ST_{i,n} - ST_{i,1}) / \Sigma_{k=1,n-1} (Y_{i,1,k} T_k).$$

Step 3: At the end of Step 2, we have the optimal solution for a batch of ΣL_p^t items. Step 3 simply adjusts all the L_i^t values so that their sum will be m .

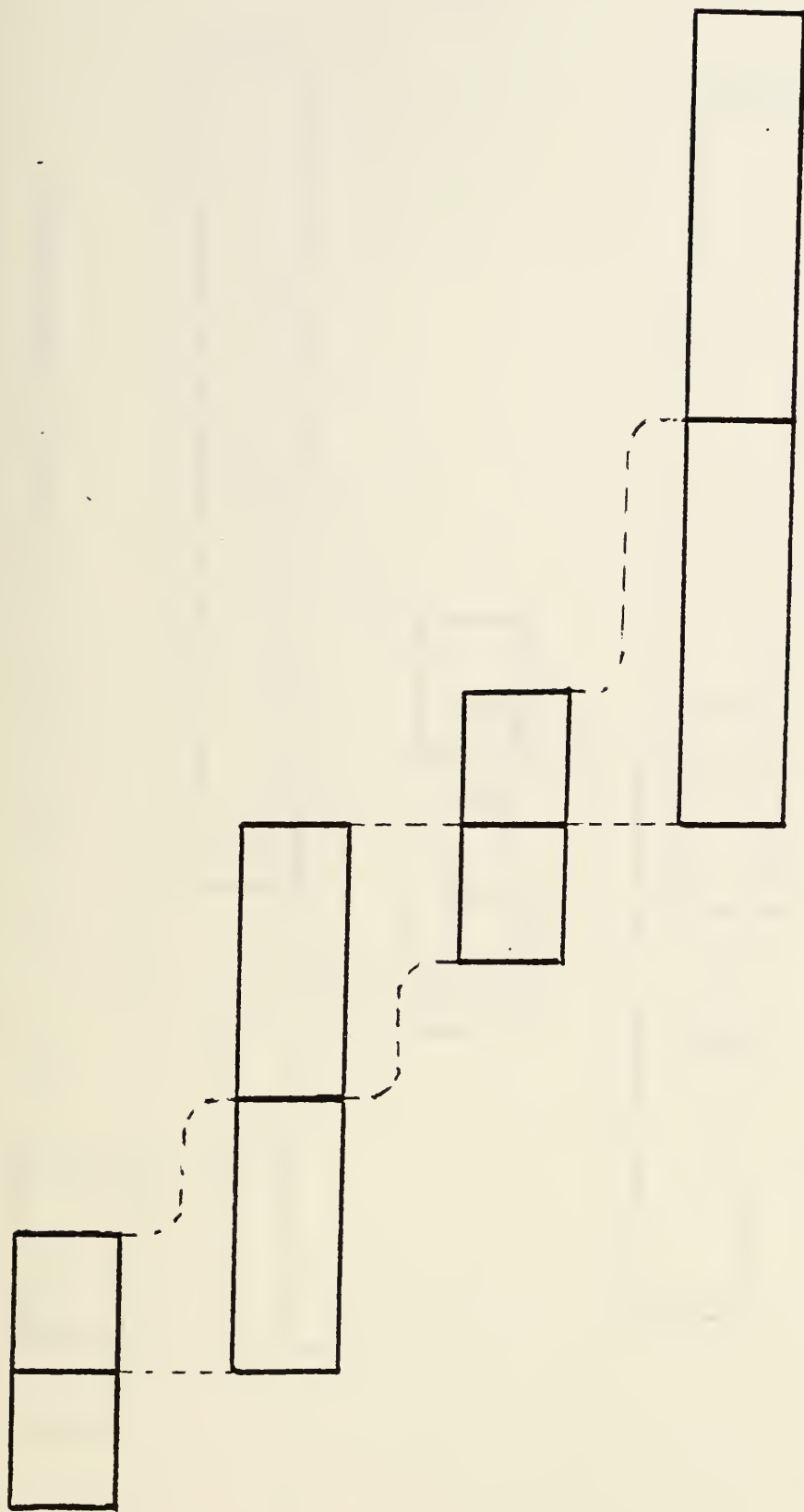


Figure 1: $m = 250$; $T_1 = 1$; $T_2 = 2$; $T_3 = 1$; $T_4 = 3$. Two equal lots. No idling.

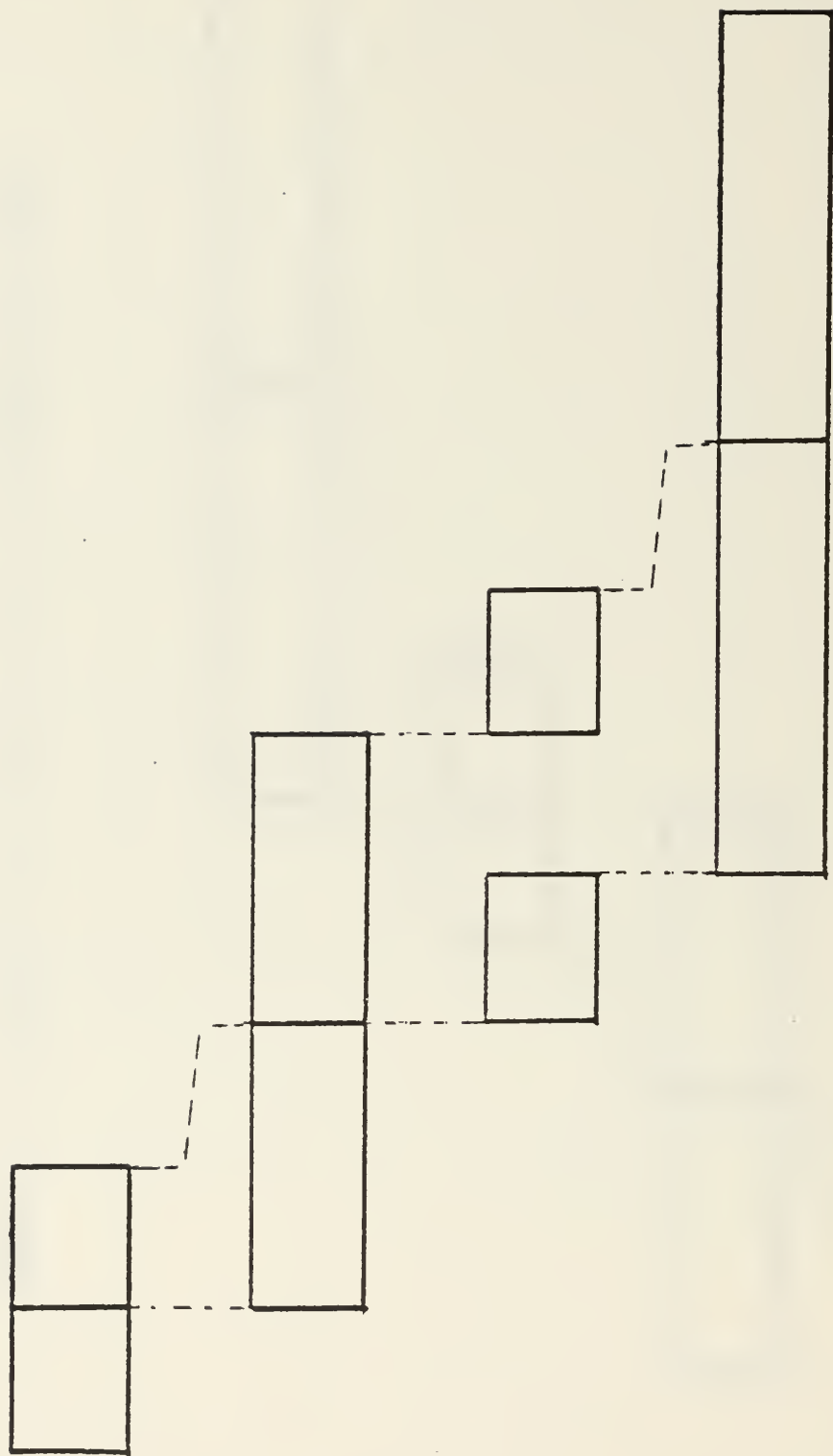


Figure 2: $m = 250$; $T_1 = 1$; $T_2 = 2$; $T_3 = 1$; $T_4 = 3$. Two equal lots. Idling allowed.

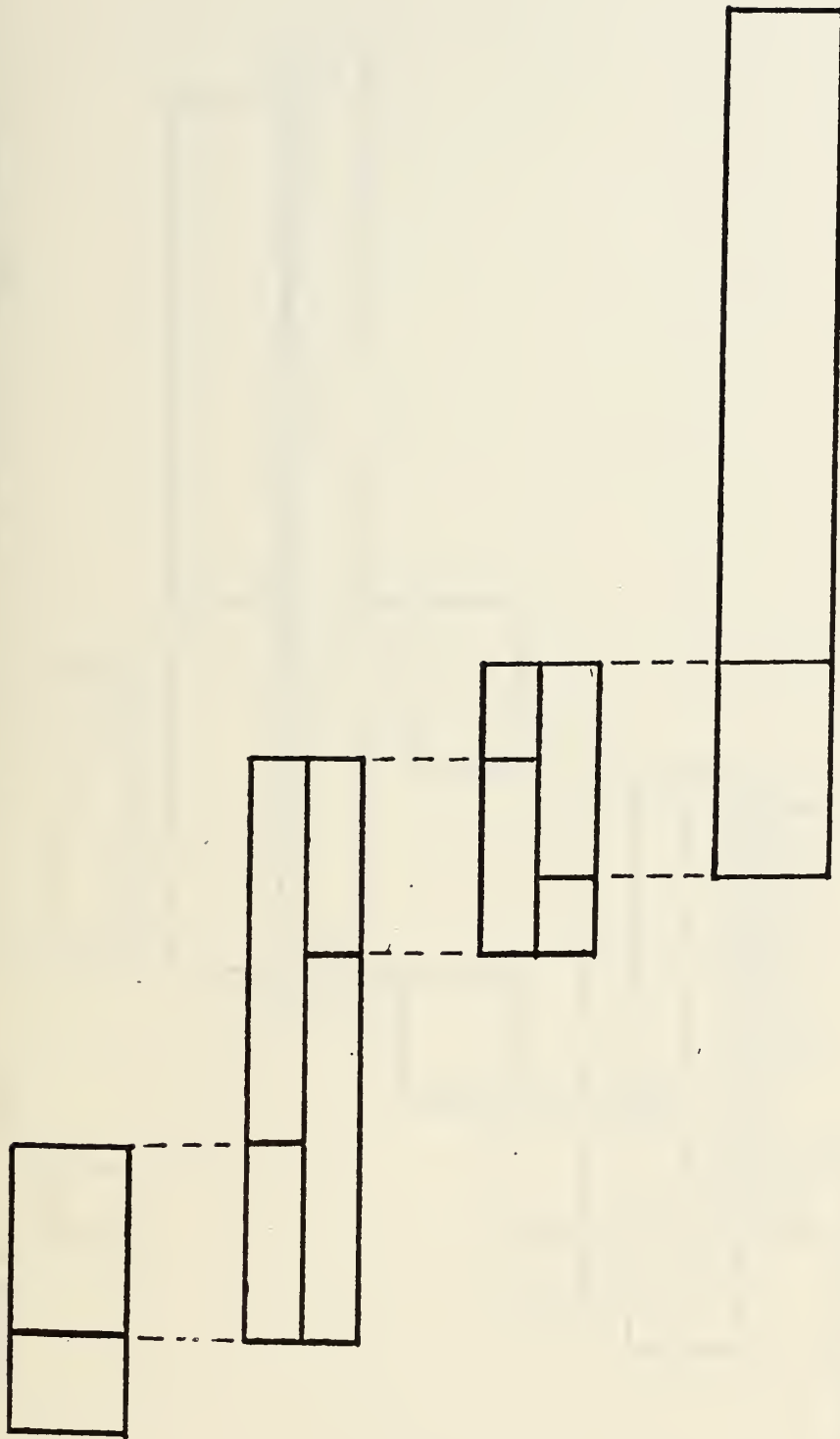


Figure 3: $m = 250$; $T_1 = 1$; $T_2 = 2$; $T_3 = 1$; $T_4 = 3$. Two varying lots. No idling.

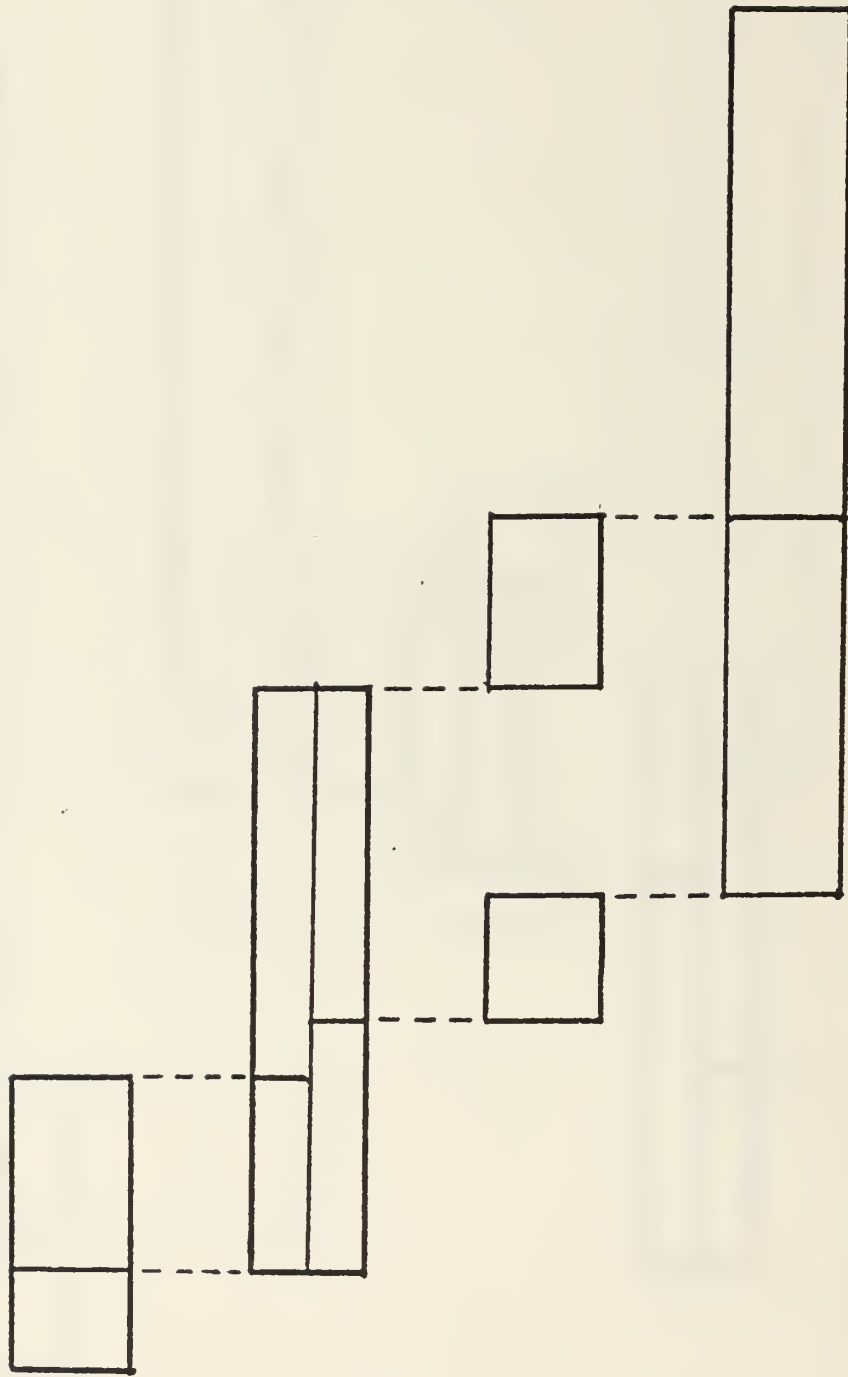


Figure 4: $m = 250$; $T_1 = 1$; $T_2 = 2$; $T_3 = 1$; $T_4 = 3$. Two varying lots. Idling allowed.

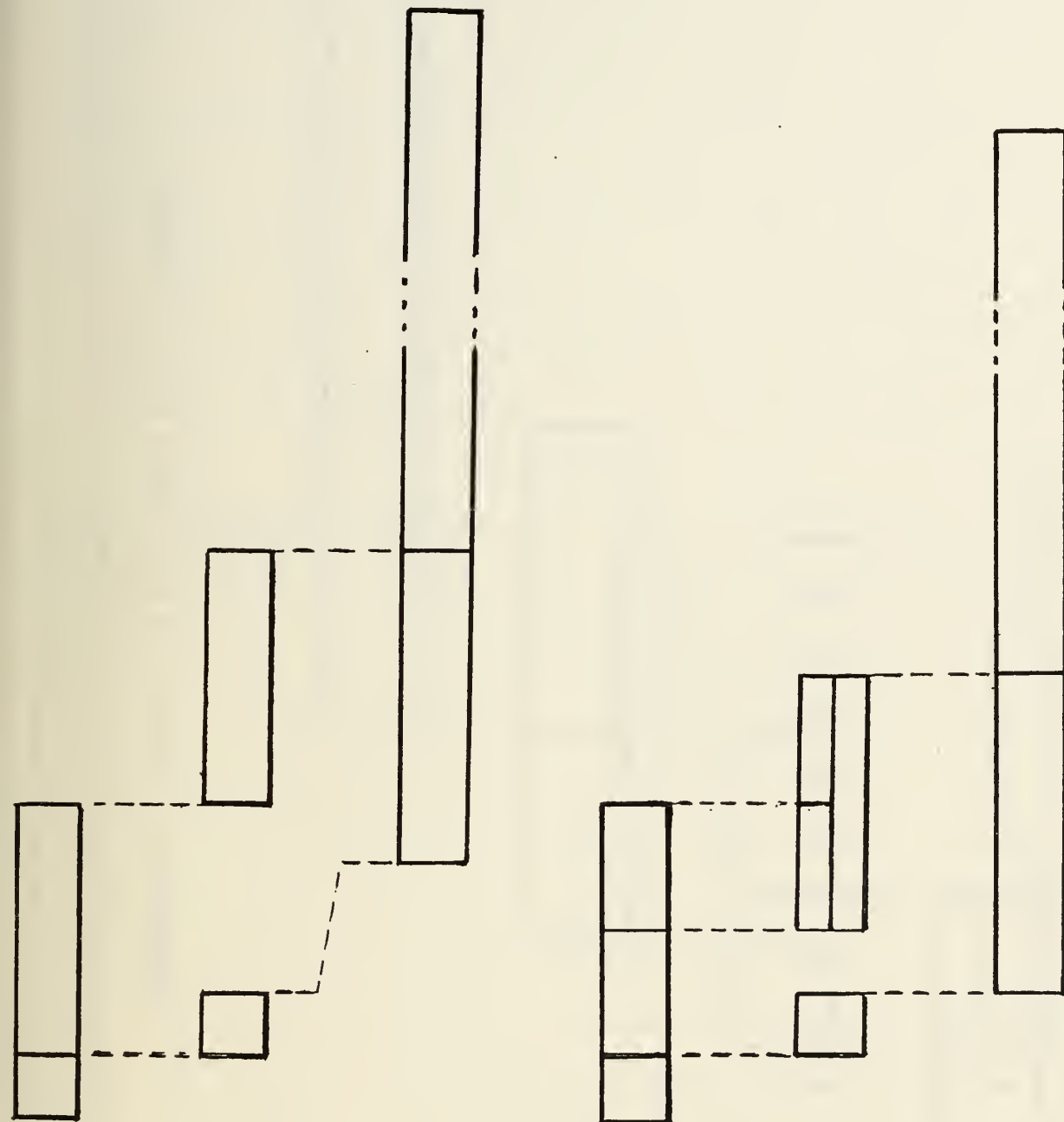


Figure 5: $m = 5$; $T_1 = 1$; $T_2 = 1$; $T_3 = 5$; $C_1 = 1$; $C_2 = 10$; $B = 23$.

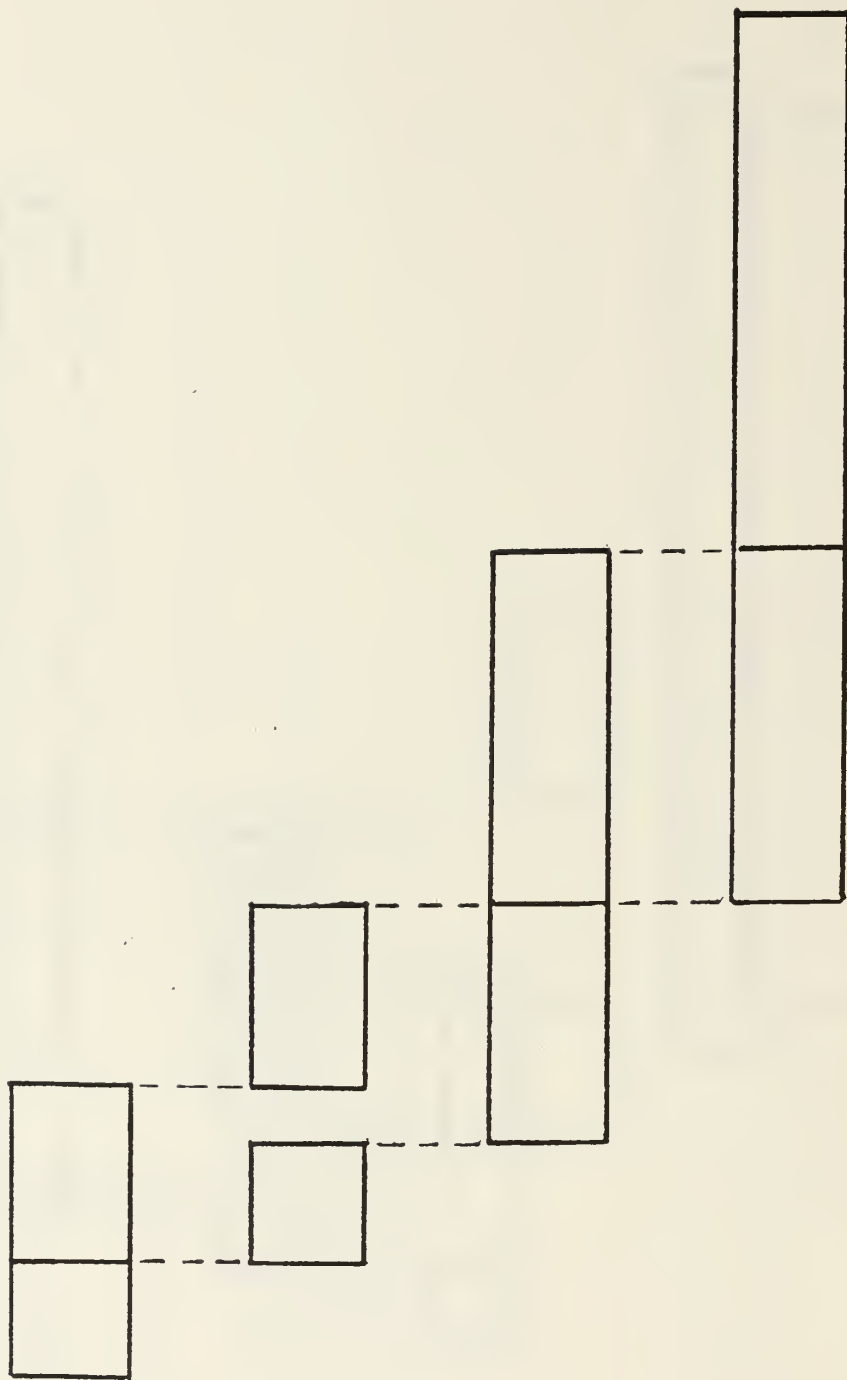


Figure 6: $m = 250$; $T_1 = 1$; $T_2 = 1$; $T_3 = 2$; $T_4 = 3$. Two lots.

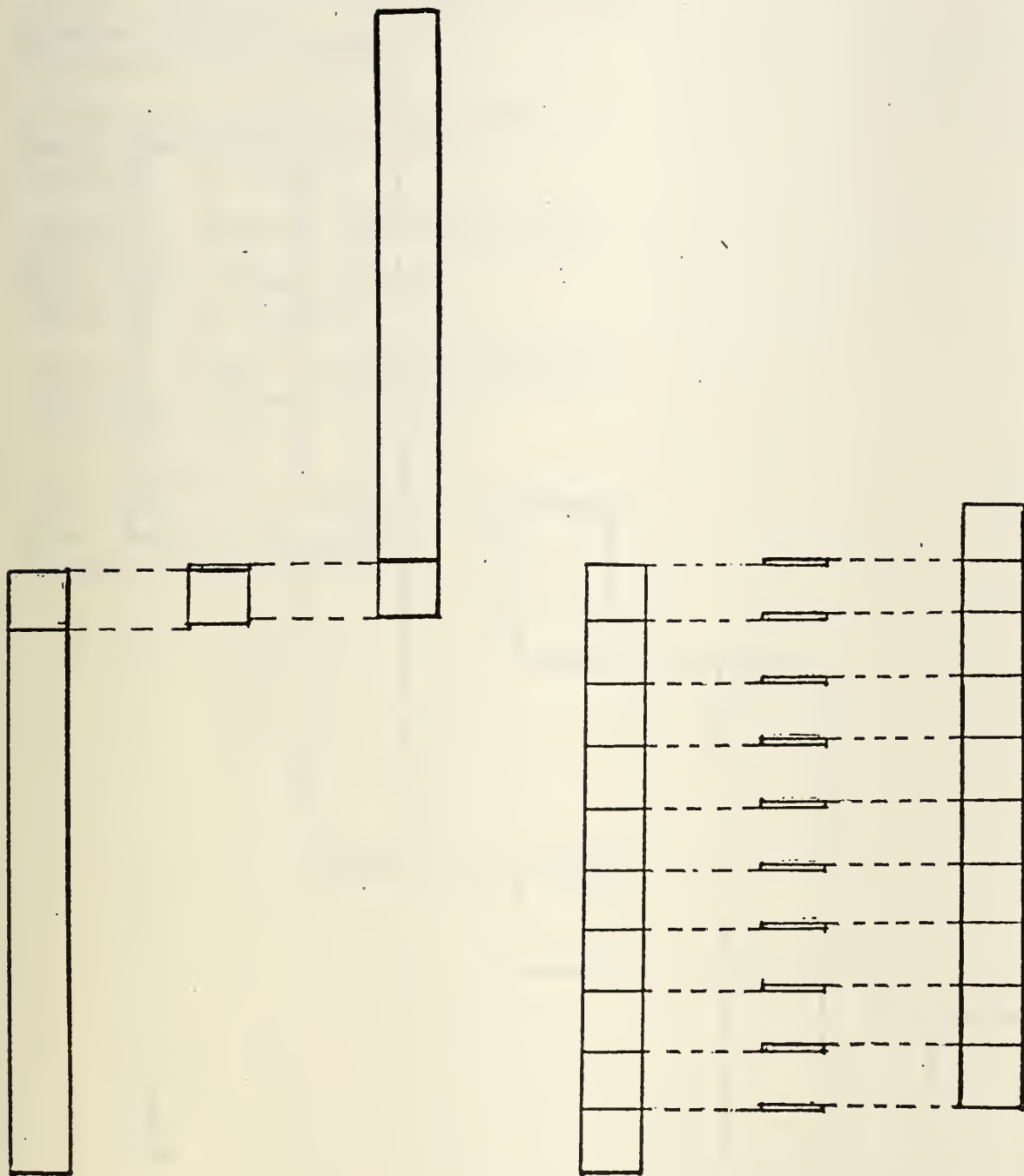


Figure 7: $m = 10$; $T_1 = 10$; $T_2 = 1$; $T_3 = 10$; $C_1 = 1$; $C_2 = 1$; $B = 20$.



Figure 8: $m = 250$; $T_1 = 1$; $T_2 = 2$; $T_3 = 1$; $T_4 = 3$. $L_1 = 100 = = >$ conflicts.

Distribution List

<u>Agency</u>	<u>No. of copies</u>
Defence Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Library, Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Dan Trietsch Code 54Tr Naval Postgraduate School Monterey, CA 93943	40

DUDLEY KNOX LIBRARY



3 2768 00338334 0